

AFIT/GSO/ENY/99M-09

ANTI-BALLISTIC MISSILE LASER PREDICTIVE  
AVOIDANCE OF SATELLITES:  
THEORY AND SOFTWARE FOR  
REAL-TIME PROCESSING AND DECONFLICTION  
OF SATELLITE EPHEMERIDES  
WITH A MOVING PLATFORM LASER

THESIS  
(Book 1 of 2)

David J. Vloedman, Captain, USAF

AFIT/GSO/ENY/99M-09

Approved for public release; distribution unlimited

DTIC QUALITY INSPECTED 2

19990409 056

The views expressed in this thesis are those of the author,  
and do not reflect the official policy or position of the  
Department of Defense, or the U.S. Government

AFIT/GSO/ENY/99M-09

ANTI-BALLISTIC MISSILE LASER PREDICTIVE AVOIDANCE OF SATELLITES:  
THEORY AND SOFTWARE FOR  
REAL-TIME PROCESSING AND DECONFLICTION OF  
SATELLITE EPHEMERIDES WITH A MOVING PLATFORM LASER  
THESIS

Presented to the Faculty of the  
Graduate School of Aeronautical and Astronautical Engineering  
of the Air Force Institute of Technology  
Air University  
Air Education and Training Command  
In Partial Fulfillment of the Requirements for the  
Degree of Master of Science in Space Operations

David J. Vloedman, B.S.

Captain, USAF


March 1999

Approved for public release; distribution unlimited

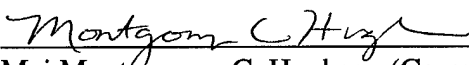
ANTI-BALLISTIC MISSILE LASER PREDICTIVE AVOIDANCE OF SATELLITES:  
THEORY AND SOFTWARE FOR  
REAL-TIME PROCESSING AND DECONFLICTION OF  
SATELLITE EPHEMERIDES WITH A MOVING PLATFORM LASER

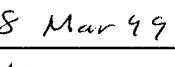
David J. Vloedman, B.S.  
Captain, USAF

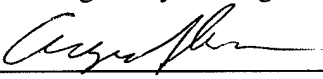
Approved:

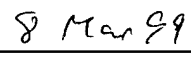
  
\_\_\_\_\_  
Dr. William E. Wiesel (Committee Chairman)

  
\_\_\_\_\_  
Date

  
\_\_\_\_\_  
Maj Montgomery C. Hughson (Committee Member)

  
\_\_\_\_\_  
Date

  
\_\_\_\_\_  
Capt Gregory Agnes (Committee Member)

  
\_\_\_\_\_  
Date

\_\_\_\_\_  
Date



## **Acknowledgments**

I would first like to thank my faculty advisor, Dr. William E. Wiesel, without whom this thesis would not exist. His knowledge and insight in the topics addressed in this thesis were critical to its formulation and conclusion in a timely manner. His time and experience were very much appreciated. I would also like to thank LtCol Glen P. Perram for his willingness to share his expertise in the area of laser propagation, which saved me many hours of independent research.

I would also like to thank my sponsors Rich Flanders and Bob Sendek working in the BMC4I IPT at Boeing's ABL Program office. Their support and patience has allowed this research to have a practical application within the ABL project. It is hoped that this thesis will benefit them as much as it has myself.

I am indebted to the many officers and civilian personnel of the Air Force Research Labs Directed Energy Program office for allowing me to attend some of the meetings concerning Predictive Avoidance and for answering my questions.

Finally, and most importantly, I would like to thank my wife, Juli, for the many hours she spent alone while I formulated this research. Her unfailing love, faithfulness and unconditional support are the truly exceptional qualities that formed the foundation this work was built on.

David J. Vloedman

## Table of Contents

	Page
Acknowledgments. . . . .	ii
List of Figures . . . . .	viii
List of Tables . . . . .	xi
Abstract . . . . .	xii
<b>I. Introduction. . . . .</b>	<b>1</b>
ABL Strategy. . . . .	1
Laser Systems. . . . .	1
The Predictive Avoidance Concept. . . . .	1
Predictive Avoidance Strategy. . . . .	3
The Goal of This Study . . . . .	4
Method of Attack . . . . .	5
<b>II. ABLPA Preprocessor Methodology . . . . .</b>	<b>7</b>
Locating the Platform . . . . .	7
Finding $\theta_g$ . . . . .	8
The ECEF Frame . . . . .	8
The ECI Frame . . . . .	8
Frame Conversion . . . . .	9
Absolute Time . . . . .	9
Sample $\theta_g$ Calculation . . . . .	11
Accuracy . . . . .	13
Finding the Platform in the ECI Frame . . . . .	14
Locating the Satellite . . . . .	15
Use of a Time Propagator . . . . .	16
Comparison of Satellite to Platform Position Vector . . . . .	17
Evaluating Time-to-Rise . . . . .	18
Ephemeris Clipping . . . . .	19
Run Intervals . . . . .	20
Ephemeris Clipping Algorithm . . . . .	20
Finding Beta . . . . .	22
Resolution of the Satellite Critical Radius . . . . .	27
Visibility of the Satellite . . . . .	29
Checking Time-to-Rise of the Satellite . . . . .	31
Preprocessor Methodology Conclusion . . . . .	32

	Page
<b>III ABLPA Main Processor Methodology</b>	<b>34</b>
Targeting the Satellite	34
The REN Frame	34
Determining Laser Position in the REN Frame	37
Position Error	38
Platform Position Error	38
Satellite Position Error	40
Missile Position Error	41
Laser Diffraction Errors	43
Laser Intrinsic Spread Error	44
Optical Diffraction and Beam Divergence	44
Other Error Considerations	46
Error Budget Consolidation	47
Finding the Current Separation Angle	50
Forecasting the Separation Angle	50
Finding the Rate of Change of the Separation Angle	51
Finding the Acceleration of the Separation Angle	53
The Forecast Method	57
Accuracy of the Forecast Method	59
Interpolation to Correct the Forecast	63
Interpolation Time Buffer	64
Interpolation Step Size	66
Main Processor Methodology Conclusion	69
<b>IV. Software Development</b>	<b>70</b>
Modularity and Testing	71
The Calculation Modules	72
The Test Modules for Each Software Library	73
The Environment	73
Sample Interfaces	74
<b>V. Final Analysis and Conclusions</b>	<b>76</b>
Software Analysis and Performance	76
Integration and Testing	77
Preprocessor Software Filtering Performance	77
Preprocessor Software Timing Performance	78
Main Processor Software Filtering Performance	78
Main Processor Software Timing Performance	79

	Page
<b>V. Final Analysis and Conclusions (cont.)</b>	
Further Study . . . . .	80
Missile Tracking . . . . .	80
Atmospheric Refraction. . . . .	81
Error Angle Determination . . . . .	82
Forecast/Interpolation Fine Tuning. . . . .	82
Software Speed and Testing . . . . .	83
Conclusion . . . . .	84
<b>Appendix A. The Software Structure . . . . .</b>	<b>85</b>
Modularity and Testing . . . . .	85
The Calculation Modules . . . . .	87
The Test Modules for Each Software Library. . . . .	87
The Environment . . . . .	88
Test Module Example . . . . .	88
Description of Code . . . . .	90
Code Listing for SGP4TestForm . . . . .	91
Error Handling . . . . .	95
<b>Appendix B. ABLPA Preprocessor Software Implementation . . . . .</b>	<b>96</b>
Preprocessor Modular Format . . . . .	97
The Core Modules . . . . .	99
Aircraft.h . . . . .	99
Satellite.h . . . . .	101
LaserConstants.h . . . . .	103
The Error Structure Library . . . . .	105
AddError . . . . .	106
GrabError . . . . .	106
CreateDisplayText . . . . .	106
The ErrorStructure.h Header File . . . . .	107
The SGP4 Support Library . . . . .	110
CallSGP4 . . . . .	111
The SGP4SupportModules.h Header File . . . . .	112
The Time Module Library . . . . .	114
ConvertCalenderToJulian . . . . .	115
ConvertJulianToCalender . . . . .	115
The TimeModules.h Header File . . . . .	116
The TLE Input Library . . . . .	119
ReadTLEFile . . . . .	120
The TLEInput.h Header File . . . . .	120

	Page
<b>Appendix B. ABLPA Preprocessor Software Implementation (cont)</b>	
The Evaluate Ephemeris Library .....	122
EvaluateEphemeris .....	123
CompareOrbit .....	125
FindThetaG .....	126
The EvaluateEphemerisModules.h Header File .....	128
The ABLPA Preprocessor .....	133
Inputs .....	134
Outputs .....	136
The PABPreprocessor.h Header File .....	137
<b>Appendix C. ABLPA Main Processor Software Implementation .....</b>	<b>139</b>
Main Processor Modular Format. ....	140
The Target Platform Library. ....	142
TargetPlatform. ....	143
The TargetPlatform.h Header File. ....	146
The Target Laser Library. ....	150
TargetLaser. ....	150
The TargetLaser.h Header File. ....	152
The Target Satellite Library. ....	155
TargetSatellite. ....	158
The TargetSatellite.h Header File. ....	159
The Find Displacement Angle Modules Library. ....	162
FindDisplacementAngles. ....	163
FindErrorAngle. ....	167
FindSeparationAngle. ....	168
The FindDisplacementAngleModules.h Header File. ....	171
The Process Satellite Library .....	178
ProcessSatellite. ....	179
InterpolateVertex. ....	182
FindDisplacementAnglesAgain. ....	184
TargetPlatformAgain. ....	187
The ProcessSatellite.h Header File. ....	191
The ABLPA Main Processor. ....	196
PAMainProcessor. ....	197
Inputs. ....	197
Outputs. ....	200
The PAMainProcessor.h Header File. ....	201

	Page
<b>Appendix D. Implementation Code</b> .....	205
Aircraft.cpp .....	205
ErrorStructure.cpp .....	209
EvaluateEphemerisModules.cpp .....	214
FindDisplacementAngleModules.cpp .....	237
PAMainProcessor.cpp .....	252
PAPreprocessor.cpp .....	259
ProcessSatellite.cpp .....	264
Satellite.cpp .....	293
SGP4SupportModules.cpp .....	299
TargetLaser.cpp .....	303
TargetPlatform.cpp .....	308
TargetSatellite.cpp .....	317
TimeModules.cpp .....	325
TLEInput.cpp .....	329
<b>Appendix E. Test Module Code</b> .....	336
EvaluateEphemerisForm.cpp. ....	336
FindDisplacementAngleForm.cpp. ....	343
MainProcessorForm.cpp. ....	350
PAPreprocessorForm.cpp. ....	355
ProcessSatelliteForm.cpp. ....	359
SGP4TestForm.cpp. ....	366
TargetPlatformForm.cpp .....	371
TargetSatelliteForm.cpp. ....	376
TestTargetLaserForm.cpp. ....	384
TestTimeForm.cpp. ....	387
TLETestForm.cpp. ....	390
TestErrorStructure.cpp (Non-Graphical Interface) .....	394
<b>Appendix F. Sample Two Line Element (TLE) File Format.</b> .....	396
<b>Bibliography</b> .....	399
<b>Vita</b> .....	400

## List of Figures

Figure	Page
2.1 Locating the Platform in the ECEF Frame . . . . .	14
2.2 Illustration of the Comparison Between $\mathbf{R}$ and $\mathbf{r}$ . . . . .	17
2.3 Ephemeris Clipping Illustration . . . . .	19
2.4 Preprocessor Spherical Geometry Illustration . . . . .	21
2.5 Northern Hemisphere Prograde Orbit Cases . . . . .	22
2.6 Northern Hemisphere Retrograde Orbit Cases . . . . .	23
2.7 Southern Hemisphere Prograde Orbit Cases . . . . .	24
2.8 Southern Hemisphere Retrograde Orbit Cases . . . . .	25
2.9 Comparison of Satellite Radius with the Critical Radius . . . . .	29
2.10 Vectors Used to Approximate Rise Time . . . . .	31
3.1 Derivation of $\rho$ in ECI Frame With Respect to the REN Frame . . . . .	35
3.2 Laser Position in the REN Frame . . . . .	37
3.3 Computing the Error Angle Contributed By Platform Position Error . . . .	39
3.4 Illustration of the Error Angle Introduced by Uncertain Missile Position . . . . .	42
3.5 Exaggerated Divergence of a Highly Focused Beam. . . . .	47
3.6 Illustration of the Separation Angle. . . . .	50
3.7 Illustration of a Satellite "Intersection" . . . . .	57
3.8 Comparing the Actual Separation Angle Encountered in a Close Approach With the Forecasted Separation Angle . . . . .	60
3.9 Illustration of Forecasted Angle Deviation From Actual Angle in a "Far Away" Satellite . . . . .	62
3.10 Typical Early Vertex Forecast with a Two Second Interpolation Buffer .	63

Figure	Page
3.11 Special Case Forecast Where Forecast Vertex Falls After Actual Vertex in Time .....	65
3.12 Interpolation With a Step Size that is Too Large .....	67
3.13 Decreasing Step Size Increases Precision .....	68
4.1 The Predictive Avoidance Software Flow .....	71
4.2 GUI Interface to the Preprocessor .....	74
4.3 GUI Interface to the Main Processor .....	75
A.1 The Predictive Avoidance Software Flow .....	86
A.2 GUI Interface to "SGP4 Support" Project .....	89
B.1 Where the ABLPA Preprocessor Fits in the Software Hierarchy .....	96
B.2 ABLPA Preprocessor Calling Tree. ....	97
B.3 Testing GUI Used to Access CallSGP4. ....	110
B.4 Graphical Interface Developed for Testing the Time Modules .....	114
B.5 Graphical Interface Developed for Testing the ReadTLEFile .....	119
B.6 Graphical Interface Used to Test EvaluateEphemeris .....	122
B.7 The Graphical Interface to the Preprocessor .....	133
C.1 Where the ABLPA Preprocessor Fits in the Software Hierarchy .....	139
C.2 ABLPA Main Processor Calling Tree .....	140
C.3 The Graphical Interface Used to Test TargetPlatform .....	143
C.4 GUI Used to Run and Test TargetLaser .....	150
C.5 Graphical Interface for TargetSatellite .....	155
C.6 GUI Used to Run and Test FindDisplacementAngles Module .....	162



Figure	Page
C.7 GUI Used to Run and Test ProcessSatellite Module . . . . .	178
C.8 The Graphical Interface Developed to Run the Main Processor . . . . .	196

## List of Tables

Table	Page
1.1 The Lasers Aboard the ABL Platform . . . . .	2
2.1 A Summary of Twelve Geometric Cases for Finding $\beta$ . . . . .	26
2.2 The True Relationship Between $\beta$ , $\alpha$ , and $i$ . . . . .	26
2.3 Resolution of Quadrant Ambiguities . . . . .	28
2.4 Possible Outcomes of Check to See if Satellite is Visible. . . . .	30
3.1 Error Budget for Predictive Avoidance Using the High Energy Laser (HEL) with a Wavelength of $1.315 \mu\text{m}$ . . . . .	48
3.2 The Meanings of the Quadratic Roots for $\Delta t$ . . . . .	59
B.1 The Six Libraries Composing the ABLPA Preprocessor. . . . .	98
C.1 The Six Libraries Composing the ABLPA Main Processor . . . . .	141
F.1 Format of Card 1 . . . . .	397
F.2 Format of Card 2 . . . . .	398

### **Abstract**

The Anti-Ballistic missile Laser (ABL) Project is committed to defense against attack from enemy-launched Theater Ballistic Missiles using an airborne laser platform to disable an enemy missile in the boost phase of launch. Wielding a laser of this power and scope requires that no collateral damage be caused by laser energy which may escape from the theater of engagement. The most likely track of such a laser would pose a significant threat to space-based assets.

The Predictive Avoidance algorithm is designed to predict the path of a given laser firing sequence, and perform real-time forecasting of, and deconfliction with, the ephemerides of a given set of satellites. The primary goal is to establish the theoretical framework of this algorithm. The secondary goal of this thesis is to develop a modular software package that can, with minor modifications, be incorporated into the fire-control system of ABL to perform real-time forecasting within given time and error budgets. This software takes the form of a Preprocessor, that filters the active satellites to determine which satellites are in view, and the Main Processor, which analyzes the satellites that are in view. The Main Processor determines whether any of the satellites in view will intersect the laser beam while it is illuminating a target.

ANTI-BALLISTIC MISSILE LASER PREDICTIVE AVOIDANCE OF SATELLITES:  
THEORY AND SOFTWARE FOR  
REAL-TIME PROCESSING AND DECONFLICTION OF  
SATELLITE EPHEMERIDES WITH A MOVING PLATFORM LASER

## **I. Introduction**

The Anti-Ballistic missile Laser (ABL) Project is committed to defense of the United States and its allies against attack from enemy-launched Theater Ballistic Missiles (TBMs). The ABL is a system which, when deployed, will be housed in a Boeing 747-400F airframe. It will fly at a cruising altitude of roughly 12.9 km (Forden, Sept 97), above most cloud cover. From this altitude, it will acquire target missiles through an active tracking system, attempting to destroy the missile in its boost stage, where it is most vulnerable.

### **1.1 ABL Strategy**

The ABL will fly above the cloud deck, and will most likely travel in an elongated figure-eight flight path. The long axis of the figure-eight will be perpendicular to the expected direction of the missile launch, to ensure the smallest focal radius to the possible targets for a given period of time.

#### **1.1.1 Laser Systems**

The ABL will actually be comprised of four independent laser systems. The first of these systems, the Active Ranging System (ARS) laser, is a frequency modulated continuous-wave carbon dioxide laser. It operates at a relatively low power (200 Watts) at a wavelength of approximately 11.15  $\mu\text{m}$ . Its purpose is solely to actively scan for a

target. The second laser, the Track Illuminator Laser (TILL) is a kilowatt-class Yb:Yag laser estimated to operate at a wavelength of approximately 1.03 micrometers. After a target has been located with the ARS, it will be illuminated with the TILL to begin active tracking of the target. The third laser system, the Beacon Illuminator Laser (BILL) is also a kilowatt-class laser system. It will be composed of a Nd:Yag laser with a wavelength of approximately 1.06 micrometers. Shortly after the TILL has begun active tracking of the Theater Ballistic Missile (TBM), the BILL will be turned on and will be used to provide real-time data to an atmospheric compensation system designed to compensate for atmospheric turbulence. After the BILL has been trained and locked onto the TBM, the final laser will be fired. This final laser, the High Energy Laser (HEL), is a Chemical Oxygen-Iodine Laser (COIL) that will operate at a wavelength of approximately 1.315 micrometers (Forden, Sept 97). Its power is estimated by the author to be between 1-3 Megawatts.

**Table 1.1 The Lasers Aboard the ABL Platform**

Device	Wavelength	Power	Pulse Type	Aperture Size
ARS	11.15 $\mu\text{m}$	200 W	FM CW	8 inches
TILL	1.03 $\mu\text{m}$	KW class	5 KHz, 50 nsec	30 cm
BILL	1.06 $\mu\text{m}$	KW class	7.5 KHz, 50 nsec	30 cm
HEL	1.315 $\mu\text{m}$	1-3 MW	CW	150 cm

### **1.1.2 The Predictive Avoidance Concept**

During the course of a mission, laser energy will almost certainly escape from the target area. The lasers of the ABL are designed for long-distance propagation, being focused in a fairly narrow beam. This means that even at great distances, escaping energy from these lasers may pose a threat to any inadvertent targets that stray into the line of fire, perhaps far downrange of the targeted missile. The exception to this rule is the ARS, which is a fairly low power laser that will attenuate quickly within the atmosphere. For this reason, the ARS is not considered when assessing a threat to inadvertent targets. The TILL, BILL and especially the HEL, however, are considered to be potentially dangerous to downrange assets. But what assets are threatened? The ABL will almost certainly be firing above the artificial horizon of the aircraft, because of the nature of the target being acquired. Therefore ground assets and aircraft are not at great risk. However, satellites are at risk. They can conceivably be at any point in the sky, and can be extremely sensitive to radiation emanating from an Earth-ward direction. Of particular interest are Low-Earth Orbiting (LEO) satellites and manned platforms that have sensors pointed towards the Earth. At LEO altitudes the lasers aboard the ABL can certainly damage these sensitive assets. The concept of Predictive Avoidance (PA) is to develop a strategy whereby the targeted missile is destroyed, while all downrange satellites are spared from potentially harmful laser radiation.

### **1.1.3 Predictive Avoidance Strategy**

The Beam Control/Fire Control (BC/FC) system within the ABL platform is a computer that controls the tracking and firing of the ABL's four lasers. Among the many

tasks of the BC/FC is to pass all commands given by the user of the system through an “engagement filter”, which, among other things, can inhibit the firing sequence if a dangerous situation (such as a satellite passing through the lazings arc) is detected (Leonard, 1998). The task, therefore, is to construct a piece of software that can monitor the locations of satellites and provide satellite/laser deconfliction information directly to the BC/FC system. This “Predictive Avoidance Software Package” (PASP) could then be run just prior to engaging a missile to ensure that no satellites are forecasted to fall within the laser’s path.

## **1.2 The Goal of This Study**

The primary thrust of this thesis is to construct a reliable real-time predictive avoidance algorithm that uses inputs as they would exist in the operational environment of the ABL platform and generates outputs that directly communicate the probability of lazings a satellite during a given mission with known mission parameters. A second goal of this study is to produce a software package that runs this predictive avoidance algorithm in real-time. This software package is designed with three conflicting (but important) objectives. The first objective is to make the software readily understandable to a person who wishes to study it in the future. This software is designed with an agreement by Boeing that it will be studied and at least partially incorporated directly into the BC/FC of the ABL platform. Therefore, to ensure a smooth incorporation into ABL, the software will be as clear and non-ambiguous as possible. The second goal is that the software be fast. It is estimated that the predictive avoidance software should not need more than 0.5 seconds to fully process a mission. Therefore, strategies must be taken to minimize processing time. The third major goal for the PA software is that it should be

modular. There are a number of methods that are used within this software that may become obsolete or deemed inaccurate by ABL engineers. While this is not an aspiration for this software, it would be foolish to not plan for this contingency. Therefore the components of this software package should be highly granular. That is, the tasks should be divided into as small of chunks as possible. It should also have clean, strongly defined interfaces between those modules. Of these three goals for the software, understandability is the most important. There are many cases within the software in which a fluid, slow, understandable implementation has been used instead of a speedier vague implementation. This is done with the understanding that the software will be reviewed at a later time, when any "slow" algorithms may be supplanted with the software engineer's choice of implementations.

### **1.3 Method of Attack**

The method that will be used here to solve the Predictive Avoidance problem is to split the task into two smaller tasks, which we shall call the Anti-Ballistic missile Laser/Predictive Avoidance (ABLPA) Preprocessor, and the ABLPA Main Processor. The ABLPA Preprocessor will handle the task of dissecting the list of satellite objects provided by US Space Command, and determining which of these objects are in view of the platform during the operational employment of the laser. The ABL Main Processor, on the other hand, will have the task of analyzing the "short" list of satellite objects in view (determined by the Preprocessor), and performing real-time calculations to compare the arc of the laser with the path of the satellite. It will be the Main Processor that is executed during the fire-sequence of ABL to determine in real-time the probability of accidentally lazng a satellite. The reason that the PA task has been split in this way is



fairly straightforward. The Main Processor must execute its task in as little time as possible, because it is run as part of the BC/FC sequence, which must rapidly acquire, track and laze the ballistic missile target. If there are fewer targets to process in the Main Processor sequence, it will execute more quickly.

## II. ABLPA Preprocessor Methodology

As mentioned previously, in order to save as much time as possible within the main processor, it is necessary to limit the number of satellites that are processed. The preprocessor filters satellite ephemerides to find only those satellites that may possibly be within the range of the laser platform for a specified time. This is accomplished in two separate tasks. In the first task, the preprocessor must locate the satellite at the current local time, and determine whether or not the satellite is currently in view. This is done with the help of an ephemeris time propagator, Satellite General Perturbations propagator (SGP4) developed by Air Force Space Command to track orbiting objects. The second task involves determining when (if ever) the satellite will come into view of the platform. This second task is only needed if the satellite is not in view. For instance, The ABLPA Preprocessor will be executed at regular intervals, but the possibility exists that a satellite may fly into view of the platform after the Preprocessor executes but before the next execution. If so, this satellite must also be included on the "short" list of objects fed to the Main Processor. These two tasks, and the methods by which they are resolved, are the focus of this chapter.

### 2.1 Locating the Platform

The first step in finding the location of the satellite with respect to the ABL platform is to determine where the platform is, and in what coordinate frame its position is known. In general, we can expect to receive the platform's position in terms of latitude ( $\delta$ ), longitude ( $\lambda$ ), and altitude ( $h$ ). This frame of reference is Earth Centered Earth Fixed (ECEF), and rotates with the Earth. Unfortunately, this method of reference does not

lend itself easily to fixing a position with respect to a satellite, whose coordinates are most often given in the Earth-Centered/Inertial (ECI) coordinate frame. Therefore a way must be found to transfer the platform's position vector from the ECEF frame to the ECI frame.

## **2.2 Finding $\theta_g$**

Let the value,  $\theta_g$ , represent the true angle between the Greenwich Meridian (that "moves" with the Earth) and the Vernal Equinox, or First Point of Aries, which is a point relatively "fixed" in the heavens. This value is important because it provides the rotation angle between the ECEF coordinate frame and the ECI frame.

### **2.2.1 The ECEF frame**

Most aircraft reference their position with respect to the Equator ( $0^\circ$  latitude), the Greenwich Meridian ( $0^\circ$  longitude), and their height above sea-level. This reference system provides a way to track location in a coordinate frame which is fixed with respect to the globe. This is the frame in which the ABL platform will likely reference its position. Both coordinate frames use the Earth's polar axis as one reference axis, and the equatorial plane as the reference plain in which the other two reference axes lie. However, the ECEF frame rotates with the Earth using the line from the center of the Earth to the Greenwich Meridian as its second reference axis.

### **2.2.2 The ECI Frame**

Because the Earth is "rotating" in space with respect to other celestial bodies, the ECEF frame becomes inconvenient to track the motions of satellites that orbit the Earth. A new frame, the ECI frame, is adopted to track these motions. This frame does not

rotate with the Earth, but rather fixes a reference axis on the Vernal Equinox, also referred to as the First Point of Aries. This provides a seemingly more absolute fixed, or “inertial”, frame with which to measure the rotation of the Earth and the motions of satellites.

### 2.2.3 Frame Conversion

As might be expected, a conversion must exist between these two frames of reference if any meaningful correlation between the motions of objects in these two reference frames is to be done.  $\theta_g$  will be used to refer to the rotation angle between these two coordinate frames, and will be used for this conversion. Fortunately, the rate of the rotation of the Earth is fairly constant, remaining relatively fixed at 1 revolution every 23 hours, 56 minutes and 4.09054 seconds (American Ephemeris and Nautical Almanac, 1980). In so stating this, I am neglecting the gradual deceleration of the Earth’s rotation, which is assumed to be negligible for the relatively short time spans with which we will be addressing here. Therefore:

$$1 \text{ sidereal day} = 23*3600 + 56*60 + 4.09054 = 86164.09054 \text{ sec} \quad (2.1)$$

The Earth rotates at the rate:

$$\frac{360^\circ}{86164.09054 \text{ sec}} = 0.004178074622 \frac{\text{deg}}{\text{sec}} = 0.0000729211585453 \frac{\text{rad}}{\text{sec}} = \dot{\theta}_g \quad (2.2)$$

### 2.2.4 Absolute Time

Now that we have a rotation rate of one coordinate frame with respect to the other, we need to specify the amount of time that transpires during our observations. Furthermore, we need to specify a starting value of  $\theta_g$  in order to propagate its value into

the future. We can obtain the former by using Modified Julian Time (MJT), which is easily mapped to Greenwich Mean Time (GMT). For brevity, I will not discuss the time mapping algorithm here, but will refer the reader to the software modules written in conjunction with this thesis. The module "TimeModules.c" performs the conversion between GMT and MJT (Numerical Recipes in C, 1990). These modules can be found at the end of this paper. We can further obtain a starting point of  $\theta_g$  by referencing the American Ephemeris and Nautical Almanac(1980) and taking any value of  $\theta_g$  which is paired with its corresponding Julian time. For example:

Date:	December 1, 1980
Julian Date:	2444574.5
Modified Julian Date:	4574.5
$\theta_g$ :	4 hrs 40 min 1.299 sec
	$=[(4*3600)+(40*60)+(1.299)] * [0.004166666667 \text{deg/sec}]$
	$= 70.0054124999 \text{ degrees}$
	$= 1.22182494234 \text{ radians}$

It is interesting to note that the position is referenced in terms of Hours, Minutes and Seconds (HMS), rather than degrees within the Almanac. Transformation between HMS and degrees is relatively straightforward. There are 24 hours in 360 degrees. Therefore:

$$1 \text{ sec} = \frac{360 \text{ deg}}{(24 \text{ hrs} * 3600 \frac{\text{sec}}{\text{hr}})} = 0.004166666667 \frac{\text{deg}}{\text{sec}} \quad (2.3)$$

Notice that we do not use the sidereal day (23 hrs, 56 min, 4.09054 sec) to translate these two measuring systems, because the arc of the angle is measured in a complete 24 hour rotation, not according to the true sidereal day.

Using this method, and the starting reference position,  $\theta_g$  can be calculated for any time in the future by propagating forward using the angular velocity of  $\theta_g$ , 0.004178074622 deg/sec. Any anomalies in the propagation, such as the gradual

deceleration of the Earth's rotation, precession, etc., can be minimized by picking a reference time closer to the propagation time, lessening the number of revolutions seen in the propagation.

### 2.2.5 Sample $\theta_g$ Calculation

In this section, the truth of the above sections will be demonstrated by showing that a value for  $\theta_g$ , propagated from a reference point, matches actual observations to a relatively high precision (American Ephemeris...1980). The date I will choose to find will be midnight, December 31<sup>st</sup>, 1980. The reference position and time will be taken as one month prior, midnight, December 1<sup>st</sup>, 1980:

#### THE REFERENCE DATE:

```

Reference Date = December 1, 1980
Reference Time = 0 hr 0 min 0 sec (midnight)
ReferenceJulianDate := 2444574.5
RefModJulianDate := ReferenceJulianDate - 2440000
RefModJulianDate = 4.5745•103

θgHours := 4
θgMinutes := 40
θgSeconds := 1.299

DegreesPerSecond :=  $\frac{360}{24 \cdot 3600}$ 
DegreesPerSecond = 4.16666666666667•10-3
RefθgDegrees := (θgHours •3600 + θgMinutes •60 + θgSeconds )•DegreesPerSecond
RefθgDegrees = 70.00541249999999

```

We will use MathCad Version 7 to propagate the angle of  $\theta_g$  which occurs at midnight on December 31, 1980, 30 days later. Notice that the reference date must be used in the calculation. Also notice that the original value for  $\theta_g$  is the total amount traversed, and

must be divided modulo 360 degrees to obtain the true value of  $\theta_g$ .

**PROPAGATION DATE :**

Propagation Date = December 31, 1980  
 Propagation Time = 0 hr 0 min 0 sec (midnight)  
 PropJulianDate := 2444604.5  
 PropModJulianDate := PropJulianDate - 2440000  
 PropModJulianDate =  $4.6045 \cdot 10^3$   
 PropagationTime := ( PropModJulianDate - RefModJulianDate )  $\cdot 24 \cdot 3600$   
 PropagationTime =  $2.592 \cdot 10^6$   
 PropagationRate :=  $\frac{360}{(23 \cdot 3600 + 56 \cdot 60 + 4.09054)}$   
 PropagationRate =  $4.178074621850468 \cdot 10^{-3}$   
 $\theta_g$ Degrees := Ref  $\theta_g$ Degrees + PropagationTime  $\cdot$  PropagationRate  
 $\theta_g$ Degrees =  $1.089957483233642 \cdot 10^4$   
 Prop  $\theta_g$ Degrees := mod (  $\theta_g$ Degrees , 360 )  
 Prop  $\theta_g$ Degrees = 99.57483233641506

The Almanac gives its observation of  $\theta_g$  for the propagation date as follows:

**THE TRUE ANGLE FOR THE PROPAGATION DATE GIVEN BY  
THE AMERICAN EPHEMERIS AND NAUTICAL ALMANAC:**

Almanac Date = December 31, 1980  
 Almanac Time = 0 hr 0 min 0 sec (midnight)  
 ReferenceJulianDate := 2444604.5  
 RefModJulianDate := ReferenceJulianDate - 2440000  
 RefModJulianDate =  $4.6045 \cdot 10^3$   
 $\theta_g$ Hours := 6  
 $\theta_g$ Minutes := 38  
 $\theta_g$ Seconds := 17.959  
 DegreesPerSecond :=  $\frac{360}{24 \cdot 3600}$   
 DegreesPerSecond =  $4.166666666666667 \cdot 10^{-3}$   
 Almanac  $\theta_g$ Degrees := (  $\theta_g$ Hours  $\cdot 3600$  +  $\theta_g$ Minutes  $\cdot 60$  +  $\theta_g$ Seconds )  $\cdot$  DegreesPerSecond  
 Almanac  $\theta_g$ Degrees = 99.57482916666666

Finally, the propagated angle must be compared with the observation from the Almanac to judge the accuracy of the algorithm.

**DIFFERENCE BETWEEN PROPAGATION ANGLE AND ALMANAC ANGLE:**

$$\theta_{\text{gDegreeDifference}} = \text{Prop } \theta_{\text{gDegrees}} - \text{Almanac } \theta_{\text{gDegrees}}$$

$$\theta_{\text{gDegreeDifference}} = 3.169748396203431 \times 10^{-6}$$

From this, we can see that in the period of 30 days,  $\theta_{\text{g}}$  can be accurately predicted to at least 0.0005% error from the true angle. This indicates a predictable accuracy below 10 meters at the Earth's surface over a time propagation of 30 days.

### **2.2.6 Accuracy**

There are some problems with predicting  $\theta_{\text{g}}$  too far into the future. Natural occurring anomalies that are difficult to model will invariably affect the true angle between the Greenwich Meridian and the Vernal Equinox. Precession, nutation and the Chandle wobble of the Earth's rotation axis, all of which cause the Earth to "wobble" on its axis rather than cleanly spin, will affect the rate of angular separation slightly. These effects are extremely difficult to model, and could be the topic of another study. They can be mitigated, however, by choosing reference date that is close to the mission date. Very little anomalous precession occurs within one week. And the closer the date, the more accurate a propagation of  $\theta_{\text{g}}$  will be. It is probably unwise to choose a reference date that is more than a couple months old, as new references are constantly being made,



and reference dates might easily be ignored, growing ever more out of date, if their update is not made a standard practice.

### 2.3 Finding the Platform in the ECI frame

All that remains for finding the coordinates of the platform in the ECI frame is to find the coordinates of the platform in the ECEF frame, and then do a single rotation using the value of  $\theta_g$  that is determined according to the methods described previously.

$R_{\oplus} + h$  = Radius of the Earth + Altitude of the Aircraft

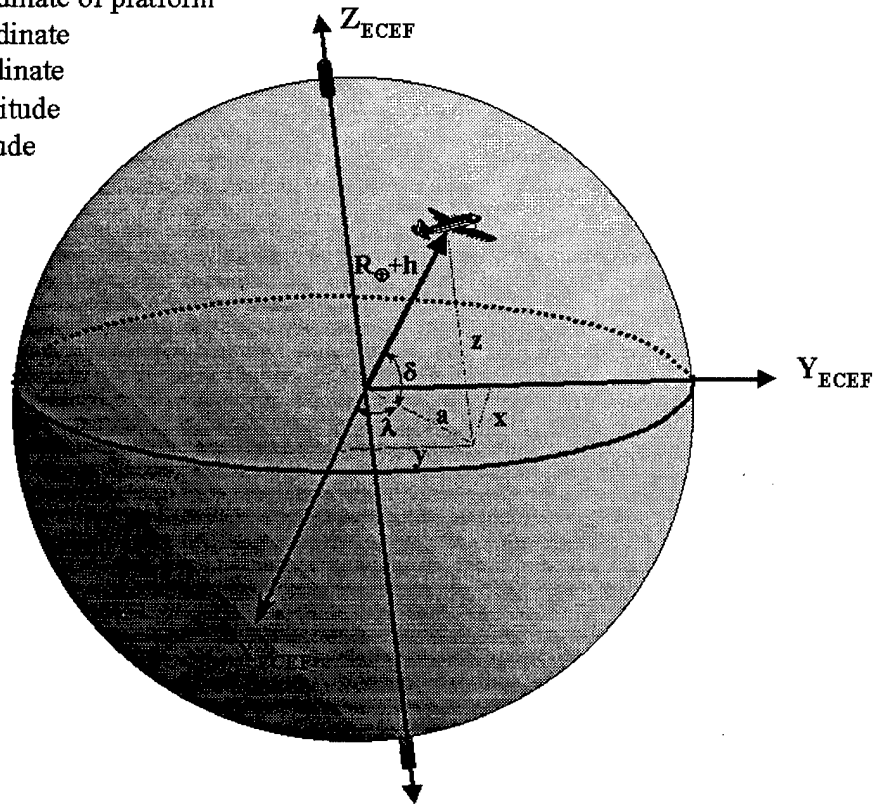
$x$  = ECEF X coordinate of platform

$y$  = ECEF Y coordinate

$z$  = ECEF Z coordinate

$\lambda$  = Degrees longitude

$\delta$  = Degrees latitude



**Figure 2.1. Locating Platform in ECEF Frame**

Figure 2.1 illustrates the conversion from latitude, longitude and altitude to X, Y and Z in the ECEF frame. From the figure it can easily be seen that:

$$a = (R_{\oplus} + h) \cos(\delta) \quad (2.4)$$

and:

$$x = (a) \cos(\lambda) \quad (2.5)$$

$$y = (a) \sin(\lambda) \quad (2.6)$$

$$z = (R_{\oplus} + h) \sin(\delta) \quad (2.7)$$

From these relations, it is seen that the coordinates of the platform in the ECEF frame are:

$$\bar{R}_{ECEF} = \begin{bmatrix} (R_{\oplus} + h) \cos(\delta) \cos(\lambda) \\ (R_{\oplus} + h) \cos(\delta) \sin(\lambda) \\ (R_{\oplus} + h) \sin(\delta) \end{bmatrix} \cdot \begin{bmatrix} \hat{X} \\ \hat{Y} \\ \hat{Z} \end{bmatrix}^T \quad (2.8)$$

To translate this position vector into the ECI coordinate frame, we use a rotation matrix about the Z axis, using  $\theta_g$  as the rotation angle:

$$\bar{R}_{ECI} = \begin{bmatrix} \cos \theta_g & -\sin \theta_g & 0 \\ \sin \theta_g & \cos \theta_g & 0 \\ 0 & 0 & 1 \end{bmatrix} \bullet \bar{R}_{ECEF} \quad (2.9)$$

From here, we also obtain the satellite's position vector by using a time propagator to determine the position at the current time. Fortunately, there are already ephemeris propagators in existence, and this project will use SGP4 Version C3.01 developed by Air Force Space Command.

## 2.4 Locating the Satellite

It should be noted here that SGP4 is used because a better propagator could not be found in the public domain. SGP4 only models general perturbations, and therefore will not be very accurate when propagating a satellite ephemeris over long periods of time. Unfortunately, this project has limited resources, and therefore cannot branch off into

intensive testing of the propagator that is used. This task could be the subject of a thesis within itself, because accuracy depends upon many factors including the type of orbit, the solar weather, the time of propagation and so on. Almost all of the errors introduced into SGP4's propagation estimates can be significantly reduced by limiting the amount of time over which the propagation occurs. This is done by ensuring that the Two-Line Element (TLE) sets used as inputs to the propagator are timely, preferably less than 24 hours old. This will ensure that the propagator on which this project relies will not have to propagate over a large amount of time, each hour of which increases the position error.

## **2.5 Use of a Time Propagator**

Both the ABLPA Preprocessor and the ABLPA Main Processor depend upon fixing the current position and velocity vectors of a satellite through the means of a time propagator. The accuracy of the propagator has a direct correlation to the effectiveness of these programs. For instance, if the satellite's position can only be estimated within a first sigma error of  $\pm 500$  kilometers, then the Main Processor will effectively see that a satellite's error cross section covers a significant field of view, making it extremely difficult to find a window in which to fire. As mentioned previously, this error can be reduced by limiting the time through which position propagation occurs. Error can also be reduced by finding a better propagator that handles special perturbations. As of the writing of this thesis, Air Force Space Command is working on the completion of just such a propagator. It is estimated that this Special Perturbations (SP) propagator will be completed by the summer of 1999. This propagator may find suitable use as a substitute for SGP4 in both the Preprocessor and the Main Processor. It may, however, prove to be too complicated for use in the Main Processor. The SP propagator, while more accurate

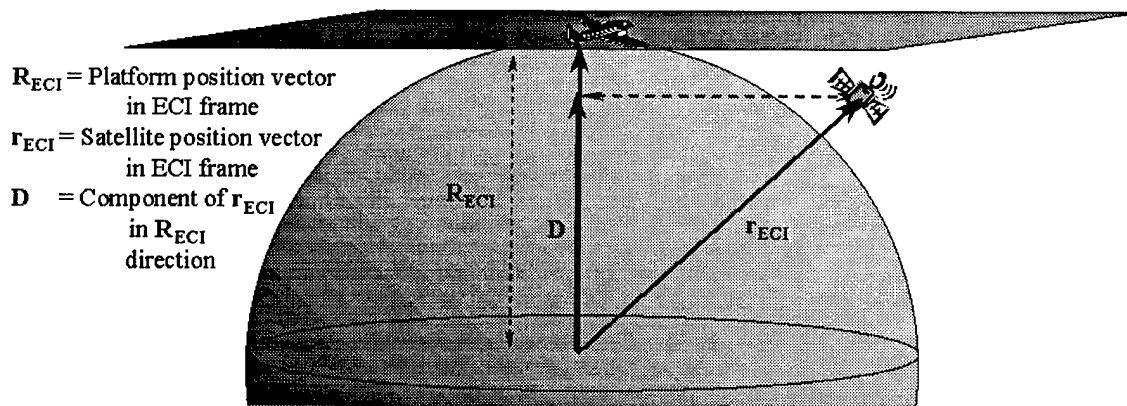
than SGP4, will almost certainly prove to be far more complicated. It will require more input parameters, and will take longer to execute. This is not a problem for the Preprocessor, as the Preprocessor is not narrowly constrained by the amount of time in which it needs to run. However, the Main Processor needs to execute quickly, in “real-time”. If the SP propagator requires an extra 0.1 seconds to run, this translates into an extra 10 seconds minimum for the Main Processor running with a 100-satellite TLE set. This is clearly unacceptable, as there cannot be a 10 second lag in the fire control system. Therefore any replacement for SGP4 will be required to pass this hurdle.

## 2.6 Comparison of Satellite to Platform Position Vector

Now that we know  $\mathbf{R}$ , the position vector of the platform in the ECI frame, and  $\mathbf{r}$ , the position vector of the satellite in the ECI frame obtained from SGP4, we can compare the two, looking to see whether the satellite crosses the artificial horizon of the platform. This can be done by finding  $\mathbf{D}$ , representing the component of  $\mathbf{r}$  in the  $\mathbf{R}$  direction:

$$\tilde{\mathbf{D}} = \tilde{\mathbf{r}}_{sat} \cdot \frac{\tilde{\mathbf{R}}_{ac}}{|\tilde{\mathbf{R}}_{ac}|} \quad (2.10)$$

It is then a simple matter to compare the magnitude of  $\mathbf{D}$  with the magnitude of  $\mathbf{R}$ . If the



**Figure 2.2.** Illustration of the Comparison Between  $\mathbf{R}$  and  $\mathbf{r}$ .

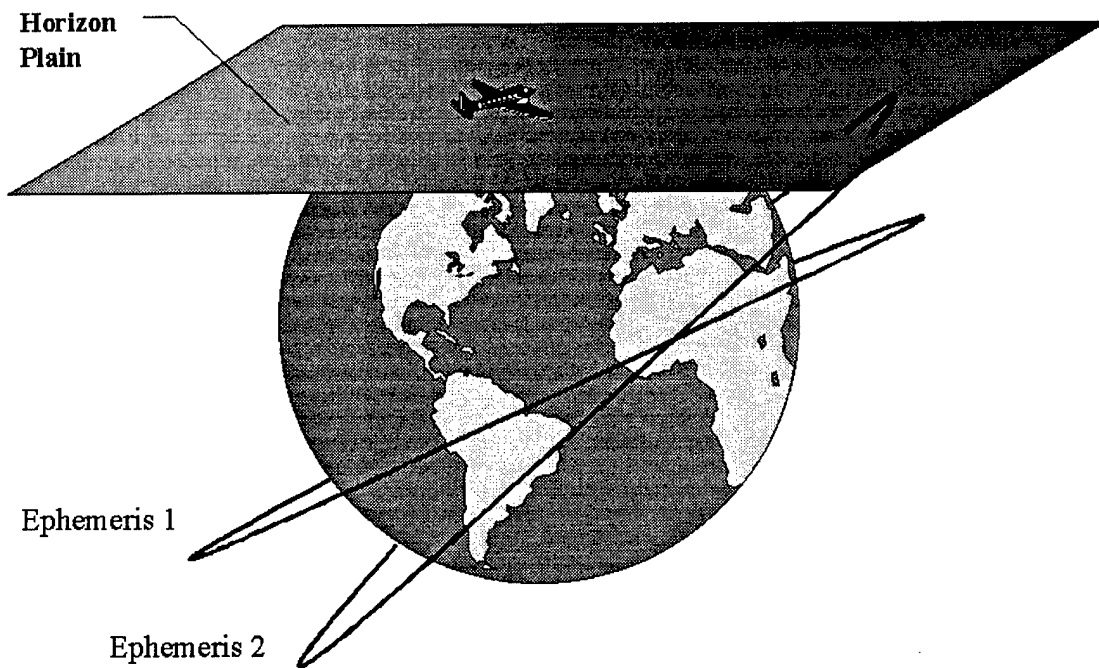
magnitude of  $\mathbf{D}$  is greater than the magnitude of  $\mathbf{R}$ , then the satellite must be above the artificial horizon of the platform, and is therefore in view.

## 2.7 Evaluating Time-to-Rise

Thus far, it has been determined whether or not the satellite is in view of the platform, but it has not been determined if the satellite might appear above the horizon at a later point in time. We are interested in how long the satellite takes to cross the horizon because there is a slice of time after the Preprocessor runs when a satellite could rise and not be noticed (until the next run of the Preprocessor). Therefore, we would like to isolate those satellites that are due to rise before the next run of the Preprocessor, so that these satellites can also be included in the list sent to the Main Processor. To do this is apparently very simple. It is fairly easy to approximate the rate of change of the magnitude of the  $\mathbf{D}$  vector discussed earlier and compare the change in  $\mathbf{D}$  to the Time Until the Next Run (TUNR). Doing so should reveal a fair estimate of whether or not the satellite is due to rise before the next run of the Preprocessor. But there is a small problem. There may be a case when the satellite *never* rises. Such is the case when the platform is at the north pole and the satellite is in an equatorial orbit. This is also the case when the platform is on the equator and 90 degrees from a polar orbiting satellite. In fact, there are many cases like this when the rate of change of  $\mathbf{D}$  is not exactly applicable to our analysis, because the satellite does not move toward or away from the platform in the short time under consideration. The solution to this problem is to weed out all ephemerides that never cross the artificial horizon plane of the platform before we evaluate the time to rise of the satellite.

## 2.8 Ephemeris Clipping

The idea of the Ephemeris Clipping Algorithm (ECA) is fairly straightforward. The ECA simply throws out all satellite ephemerides that do not cross the artificial horizon plane *in the current orbit*. Thus, in the case of Figure 2.3, Ephemeris 1 would be discarded while Ephemeris 2 will be further evaluated.



**Figure 2.3. Ephemeris Clipping Illustration**

It is important to note that this algorithm only evaluates a satellite in its current orbit, and does not propagate the ephemeris in time. Therefore the orbital elements that are used to evaluate this orbit must be propagated (also using SGP4) to the current time. It is therefore assumed that the ephemeris does not change significantly in the time between runs of the Preprocessor. Although precession of the Earth may change the attitude of the platform with respect to the satellite slightly in this time, the variations will be slight if the time between runs is kept below 5 minutes.

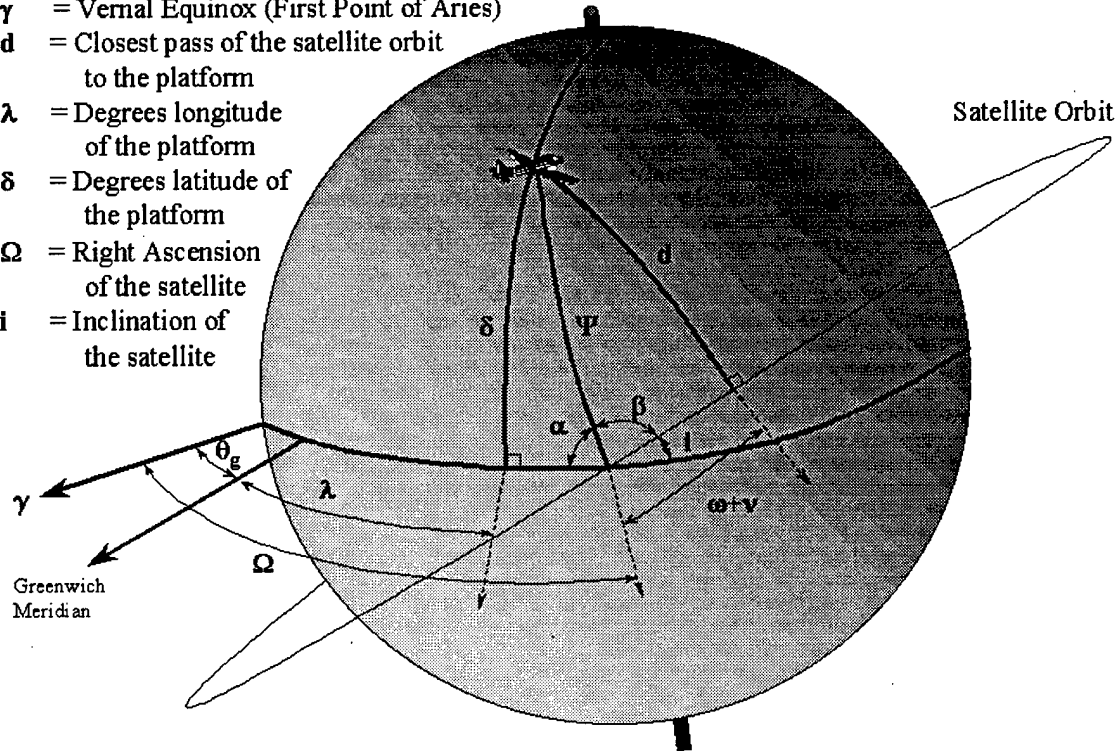
### **2.8.1 Run Intervals**

The preprocessor is designed to run at given intervals, processing a new set of “at-risk” satellites during each interval. The interval time should be long enough so that the preprocessor can easily process all ephemerides which are fed to it, and still short enough so that it can eliminate a majority of the satellites which will not be seen during the interval. For example, picking an interval of one second may not be wise, because the preprocessor may not be able to run through the enormous amount of data in that time frame. Furthermore, picking an interval time of 90 minutes may not be very beneficial, because many Low-Earth Orbiting (LEO) satellites have periods of roughly 90 minutes. Therefore if a satellite is still an hour away from being visible, it will still have to be looked at by the Main Processor during the fire sequence. For the purposes of this study, 5 minutes will be used as a nominal time interval. Thus, the preprocessor will generally use the current time as the start of the processing time interval, and the current time plus 5 minutes as the end of the processing interval. This will mean that the preprocessor must be run every 5 minutes at a minimum, but may be run more often, if desired.

### **2.8.2 Ephemeris Clipping Algorithm**

The geometry of the plane-clipping problem is illustrated in Figure 2.4. For each satellite being evaluated, we are given as inputs the orbital elements of the satellite and the time interval we are evaluating. Our goal is to find whether or not the satellite crosses the platform’s plane of view. As a beginning, the minimum distance  $d$ , between the platform and the satellite orbit needs to be determined. This is logically the line that forms a right angle with the orbit plane when drawn from the platform to the orbit plane.

- $\theta_g$  = Degrees between Greenwich Meridian and Vernal Equinox  
 $\omega+v$  = Argument of Perigee + the True Anomaly of the satellite  
 $\Psi$  = Angle between ascending node of satellite and the platform  
 $\gamma$  = Vernal Equinox (First Point of Aries)  
 $d$  = Closest pass of the satellite orbit to the platform  
 $\lambda$  = Degrees longitude of the platform  
 $\delta$  = Degrees latitude of the platform  
 $\Omega$  = Right Ascension of the satellite  
 $i$  = Inclination of the satellite



**Figure 2.4. Preprocessor Spherical Geometry Illustration**

The two spherical triangles illustrated in Figure 2.4 are used to determine  $d$  in degrees.

From the figure, it can be seen that  $\Psi$  represents the spherical angle from the ephemeris right ascension to the platform. Using the spherical right triangle formulae the length of  $\Psi$  can be determined by:

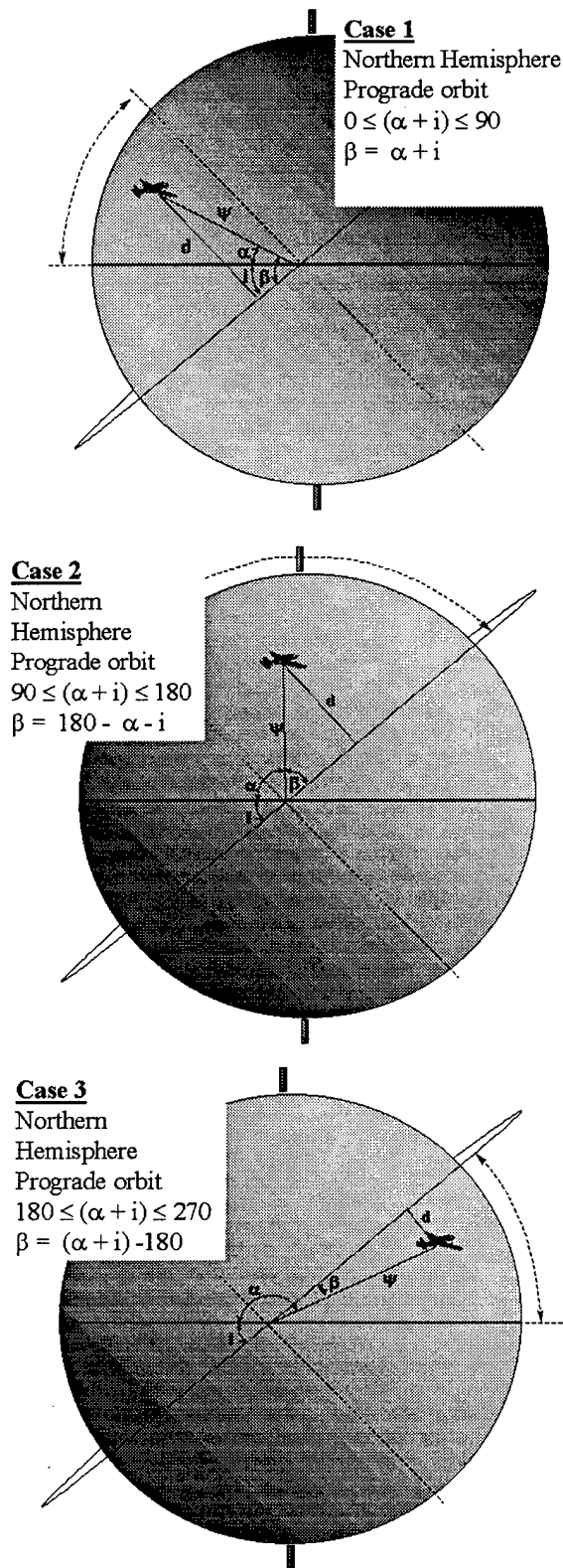
$$\cos(\Psi) = \cos(\delta) \cos(\Omega - \theta_g - \lambda) \quad (2.11)$$

Once  $\Psi$  has been found,  $\alpha$ , or the angle between  $\Psi$  and the equator, is easily found with the equation:

$$\cos(\alpha) = \tan(\Omega - \theta_g - \lambda) \cot(\Psi) \quad (2.12)$$

The angle between  $\Psi$  and the ephemeris propagation direction is denoted as  $\beta$ . As shown in the following examples, the method for finding  $\beta$  will differ depending upon location.





**Figure 2.5. Northern Hemisphere, Prograde Orbit Cases**

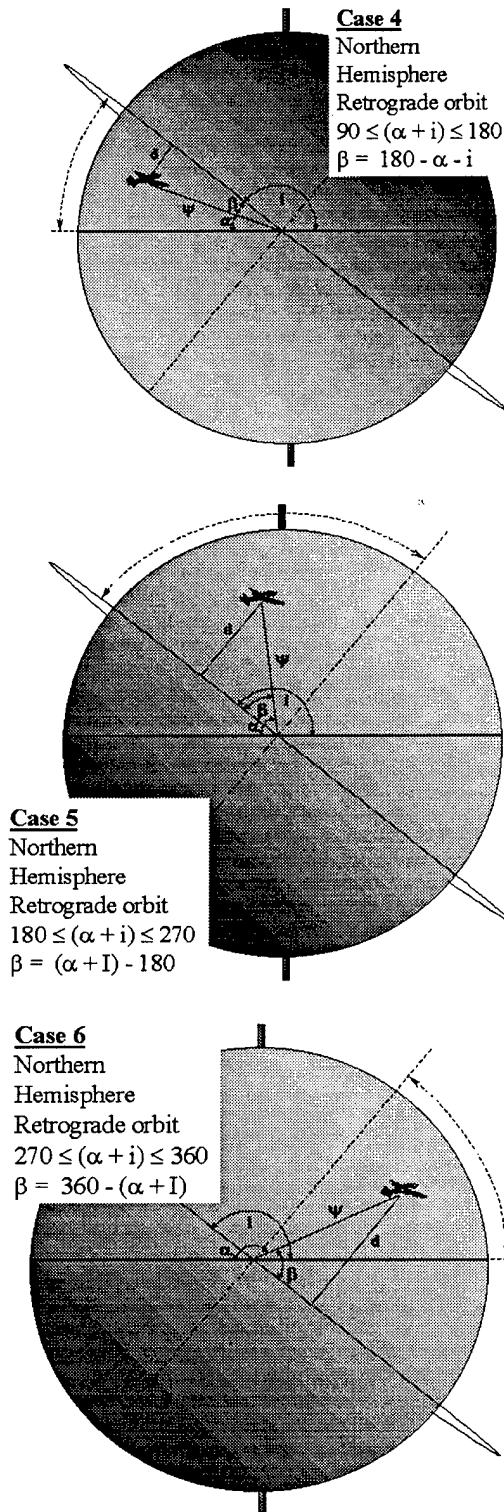
### 2.8.3 Finding Beta

Unfortunately,  $\beta$  cannot be found with a single equation. As a matter of fact, it must be found by looking at twelve individual geometric cases which serve to describe the relationship between  $\beta$ ,  $\alpha$ , and  $i$ .

The relationship is not a trivial one. It is governed by the hemisphere of the Earth (northern or southern) in which the platform lies, the slope of the orbit (prograde or retrograde), and the position of the platform with respect to the ephemeris plane.

The first three cases deal with the possibility that the platform is in the northern hemisphere, and the inclination of the satellite orbit is less than 90 degrees (a prograde orbit). As the graphical depiction shows,  $\beta$  can be three different values, depending upon where the platform lies with respect to the ephemeris path. For instance, in **Case 1**,

$$\begin{aligned} 0 \leq (\alpha + \beta) \leq 90 \\ \beta = \alpha + i \end{aligned} \quad (2.13)$$



Likewise in **Case 2**:

$$\begin{aligned} 90 \leq (\alpha + i) \leq 180 \\ \beta = 180 - \alpha - i \end{aligned} \quad (2.14)$$

and in **Case 3**:

$$\begin{aligned} 180 \leq (\alpha + i) \leq 270 \\ \beta = \alpha + i - 180 \end{aligned} \quad (2.15)$$

While cases 1 to 3 deal with a prograde orbit, cases 4 to 6 describe the relationship between a platform in the northern hemisphere and a satellite with a retrograde orbit. One might be tempted to jump to the conclusion that finding  $\beta$  would be the same as in cases 1 to 3, however, as can be seen in Figure 2.6, the relationship is not the same. In **Case 4**, where the platform is located below the retrograde orbit and above the equator:

$$\begin{aligned} 90 \leq (\alpha + i) \leq 180 \\ \beta = 180 - \alpha - i \end{aligned} \quad (2.16)$$

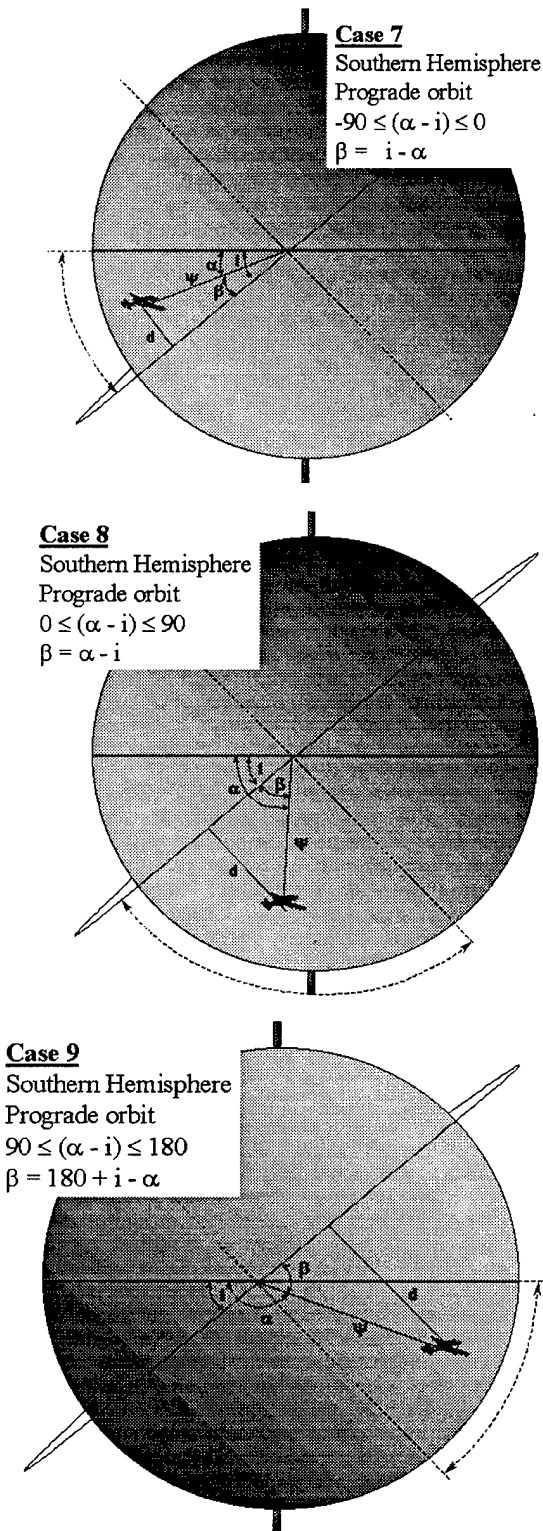
Similarly, in **Case 5**:

$$\begin{aligned} 180 \leq (\alpha + i) \leq 270 \\ \beta = \alpha + i - 180 \end{aligned} \quad (2.17)$$

And in **Case 6**:

$$\begin{aligned} 270 \leq (\alpha + i) \leq 360 \\ \beta = 360 - \alpha - i \end{aligned} \quad (2.18)$$

**Figure 2.6. Northern Hemisphere, Retrograde Orbit Cases**



**Figure 2.7. Southern Hemisphere, Prograde Orbit Cases**

Just as there are six possible configurations in the northern hemisphere, there are also six configurations in the southern hemisphere, three for prograde orbits and three for retrograde. Cases 7 to 9 illustrate the possible scenarios for a platform in the southern hemisphere with a prograde satellite orbit. For Case 7:

$$\begin{aligned} -90 \leq (\alpha - i) \leq 0 \\ \beta = i - \alpha \end{aligned} \quad (2.19)$$

For Case 8:

$$\begin{aligned} 0 \leq (\alpha - i) \leq 90 \\ \beta = \alpha - i \end{aligned} \quad (2.20)$$

And for Case 9:

$$\begin{aligned} 90 \leq (\alpha - i) \leq 180 \\ \beta = 180 + i - \alpha \end{aligned} \quad (2.21)$$

The last three cases illustrate the possibilities in the southern hemisphere combined with a retrograde orbit. For Case 10:

$$\begin{aligned} -180 \leq (\alpha - i) \leq -90 \\ \beta = 180 + \alpha - i \end{aligned} \quad (2.22)$$

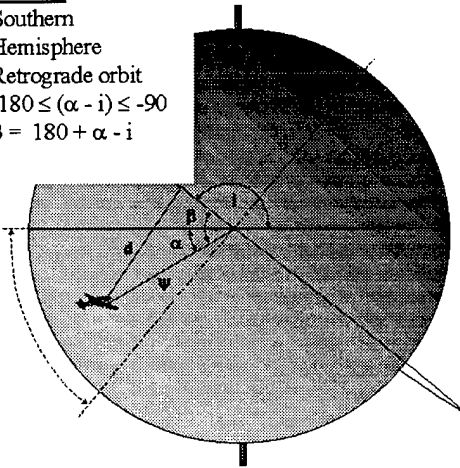
For Case 11:

$$\begin{aligned} -90 \leq (\alpha - i) \leq 0 \\ \beta = i - \alpha \end{aligned} \quad (2.23)$$

**Case 10**Southern Hemisphere  
Retrograde orbit

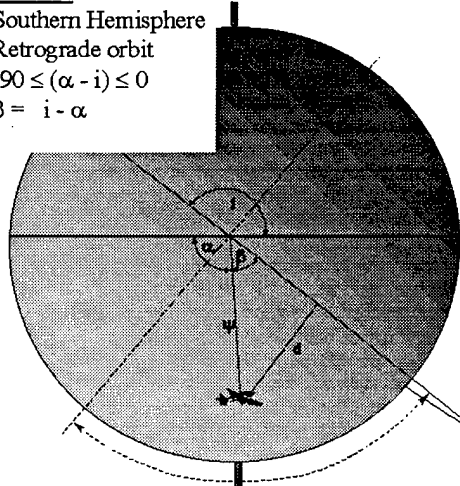
$$-180 \leq (\alpha - i) \leq -90$$

$$\beta = 180 + \alpha - i$$

**Case 11**Southern Hemisphere  
Retrograde orbit

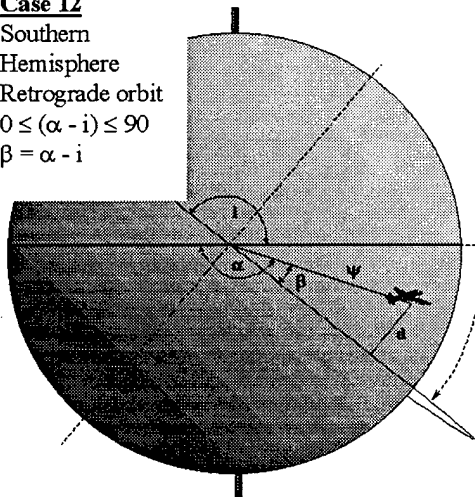
$$-90 \leq (\alpha - i) \leq 0$$

$$\beta = i - \alpha$$

**Case 12**Southern Hemisphere  
Retrograde orbit

$$0 \leq (\alpha - i) \leq 90$$

$$\beta = \alpha - i$$

And finally, for **Case 12**:

$$0 \leq (\alpha - i) \leq 90$$

$$\beta = \alpha - i \quad (2.24)$$

The reader might notice a few patterns from the twelve possible scenarios presented here. First, notice that all cases can be described by  $(\alpha + i)$  in the northern hemisphere, and  $(\alpha - i)$  in the southern hemisphere. Second, despite the fact that there are twelve possible position scenarios with respect to hemisphere, platform and satellite orbit, there appears to be only eight independent equations for determining  $\beta$ . Table 2.1 shows all equations for  $\beta$  listed together. Notice that some of the cases have the same equation for determining  $\beta$ . From this table it becomes readily apparent the true relationship between  $\alpha$ ,  $i$ , and  $\beta$ . This final relationship is illustrated in Table 2.2. It is these equations in Table 2.2 that are used to determine  $\beta$  in the preprocessor software.

**Figure 2.8. Southern Hemisphere, Retrograde Orbit Cases**

**Table 2.1. A Summary of Twelve Geometric Cases for Finding  $\beta$**

Hemisphere	Orbit Type	$(\alpha+i)$	$(\alpha-i)$	$\beta$
North	Prograde	$0 \leq (\alpha+i) \leq 90$	N/A	$\beta = \alpha + i$
North	Prograde	$90 \leq (\alpha+i) \leq 180$	N/A	$\beta = 180 - \alpha - i$
North	Prograde	$180 \leq (\alpha+i) \leq 270$	N/A	$\beta = \alpha + i - 180$
North	Retrograde	$90 \leq (\alpha+i) \leq 180$	N/A	$\beta = 180 - \alpha - i$
North	Retrograde	$180 \leq (\alpha+i) \leq 270$	N/A	$\beta = \alpha + i - 180$
North	Retrograde	$270 \leq (\alpha+i) \leq 360$	N/A	$\beta = 360 - \alpha - i$
South	Prograde	N/A	$-90 \leq (\alpha-i) \leq 0$	$\beta = i - \alpha$
South	Prograde	N/A	$0 \leq (\alpha-i) \leq 90$	$\beta = \alpha - i$
South	Prograde	N/A	$90 \leq (\alpha-i) \leq 180$	$\beta = 180 + i - \alpha$
South	Retrograde	N/A	$-180 \leq (\alpha-i) \leq -90$	$\beta = 180 + \alpha - i$
South	Retrograde	N/A	$-90 \leq (\alpha-i) \leq 0$	$\beta = i - \alpha$
South	Retrograde	N/A	$0 \leq (\alpha-i) \leq 90$	$\beta = \alpha - i$

**Table 2.2. The True Relationship Between  $\beta$ ,  $\alpha$ , and  $i$**

Hemisphere	$(\alpha+i)$	$(\alpha-i)$	$\beta$
North	$0 \leq (\alpha+i) \leq 90$	N/A	$\beta = \alpha + i$
North	$90 \leq (\alpha+i) \leq 180$	N/A	$\beta = 180 - \alpha - i$
North	$180 \leq (\alpha+i) \leq 270$	N/A	$\beta = \alpha + i - 180$
North	$270 \leq (\alpha+i) \leq 360$	N/A	$\beta = 360 - \alpha - i$
South	N/A	$-180 \leq (\alpha-i) \leq -90$	$\beta = 180 + \alpha - i$
South	N/A	$-90 \leq (\alpha-i) \leq 0$	$\beta = i - \alpha$
South	N/A	$0 \leq (\alpha-i) \leq 90$	$\beta = \alpha - i$
South	N/A	$90 \leq (\alpha-i) \leq 180$	$\beta = 180 + i - \alpha$

The final distillation of relationships between  $\beta$ ,  $\alpha$ , and  $i$  shown in Table 2.2 no longer uses the orbit type as a reference, because the emphasis is more rigorously based upon the combination of  $\alpha+i$  and  $\alpha-i$  on  $\beta$ .

#### 2.8.4 Resolution of the Satellite Critical Radius

Knowing  $\beta$  and  $\Psi$  allows  $d$  to be calculated by following another rule for spherical triangles:

$$\sin(d) = \sin(\beta) \sin(\Psi) \quad (2.25)$$

The true anomaly  $v$  can be extracted in a similar fashion, assuming that the argument of perigee,  $\omega$ , the minimal distance,  $d$ , the angle,  $\beta$ , and the distance,  $\Psi$ , in degrees are all known:

$$\sin(v + \omega) = \tan(d) \cot(\beta) \quad (2.26)$$

$$\cos(v + \omega) = \cos(\Psi) / \cos(d) \quad (2.27)$$

$$v = (v + \omega) - \omega \quad (2.28)$$

Notice that finding both the sine and cosine in Equations 2.26 and 2.27 allow a way to resolve quadrant errors which would arise from using either the sine or the cosine alone. Unfortunately,  $v+\omega$  can be anywhere from 0 to 360 degrees. The sine or cosine alone can only pinpoint 0 to 180 degrees. Although the method for dealing with quadrant errors is straightforward, the method used for this study is listed here for clarity in Table 2.3.

From the true anomaly, the scalar radius,  $r_{sat}$ , of the satellite orbit at the closest point may be determined. The value  $r_{sat}$  represents the distance of the satellite from the center of the Earth at the point where the satellite is closest to the platform. Assuming that the

**Table 2.3. Resolution of Quadrant Ambiguities**

GIVEN	$\sin(x) = y$ $\cos(x) = z$
IF $\sin^{-1}(y)$ is positive or 0 AND $\cos^{-1}(z)$ is positive or 0 THEN	$x = \sin^{-1}(y)$
IF $\sin^{-1}(y)$ is positive or 0 AND $\cos^{-1}(z)$ is negative THEN	$x = 180^\circ - \sin^{-1}(y)$
IF $\sin^{-1}(y)$ is negative AND $\cos^{-1}(z)$ is positive or 0 THEN	$x = 360^\circ - \sin^{-1}(y)$
IF $\sin^{-1}(y)$ is negative AND $\cos^{-1}(z)$ is negative THEN	$x = 180^\circ + \sin^{-1}(y)$

eccentricity,  $e$ , and the semi-major axis,  $a$ , are known,  $r_{\text{sat}}$  can be found by applying the equation for a conic section:

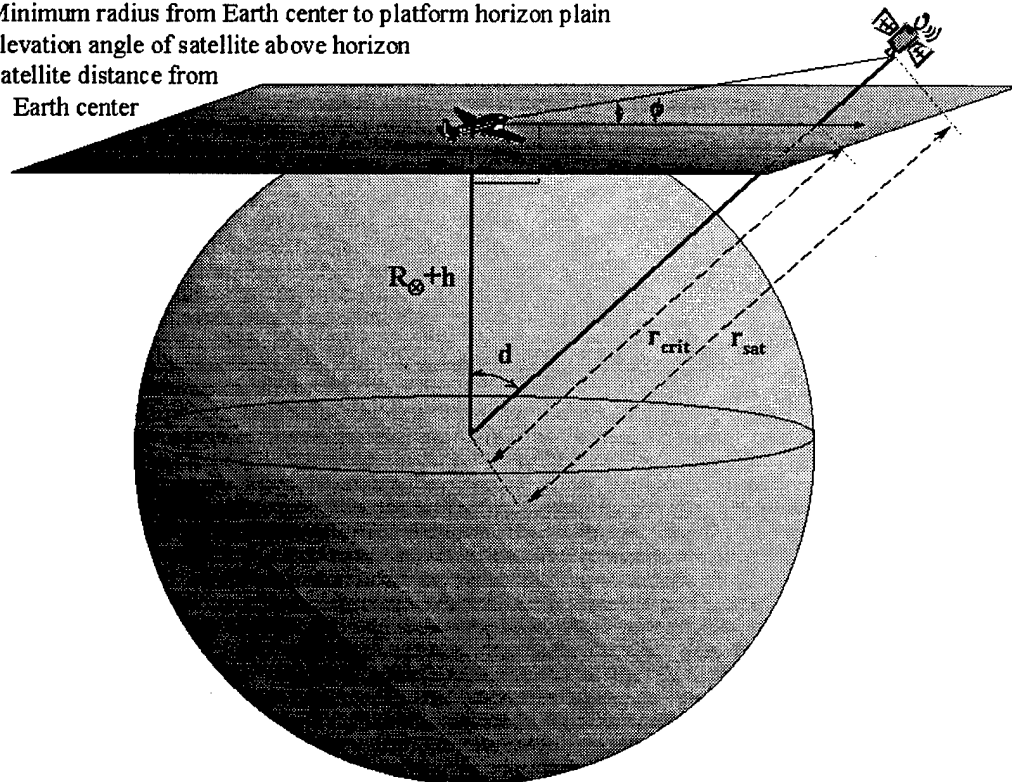
$$r_{\text{sat}} = a \frac{(1 - e^2)}{1 + e \cos(v)} \quad (2.29)$$

Now that  $r_{\text{sat}}$  is known, it must be compared to the critical radius,  $r_{\text{crit}}$ , to determine whether the satellite crosses the platform viewing horizon. From Figure 2.9 it can be seen that the critical radius is defined by:

$$r_{\text{crit}} = (R_{\oplus} + h) / \cos(d) \quad (2.30)$$

If the satellite orbit radius measured from the center of the Earth is greater than the critical radius for the orbit, then the satellite will have to be further processed by the preprocessor in order to determine whether the ephemeris reaches the platform horizon

$R_{\oplus}+h$  = Radius of the Earth + the height of the platform  
 $d$  = Degree distance of closest approach to the satellite  
 $r_{crit}$  = Minimum radius from Earth center to platform horizon plain  
 $\phi$  = Elevation angle of satellite above horizon  
 $r_{sat}$  = Satellite distance from Earth center



**Figure 2.9. Comparison of Satellite Radius with the Critical Radius**

plane during the specified time interval. If the critical radius is larger, however, the ephemeris can be discarded as out of range.

## 2.9 Visibility of the Satellite

To recap, the intention of this preprocessor analysis is to find all satellites on the input TLE file that will be visible to the platform from the current time until the time that the preprocessor is next run. The preceding discussion has shown that we can determine whether the satellite is in view of the platform, and whether the satellite's ephemeris is in view. This leaves us with four possible cases concerning the visibility of the satellite from the platform. These four cases are summarized in Table 2.4. The first case is that we have found that the satellite is currently in view, in which case our analysis stops and



this satellite is included in the input TLE file to the Main Processor. The second case is that neither the satellite nor its ephemeris is visible before the next run of the preprocessor, in which case the satellite is thrown out because it is always out of range for the time window of our analysis. The third and fourth cases require a bit more explanation. In the third case, the satellite is not in view, but its ephemeris is in view. As stated previously, this presents a dilemma whereby the satellite could move into view of the platform before the Time Until Next Run (TUNR) of the preprocessor. In this case, the satellite could be in view during a window of time after the preprocessor is run, but it may not be included in the list of satellites sent to the main processor until the next run of the preprocessor has been made.

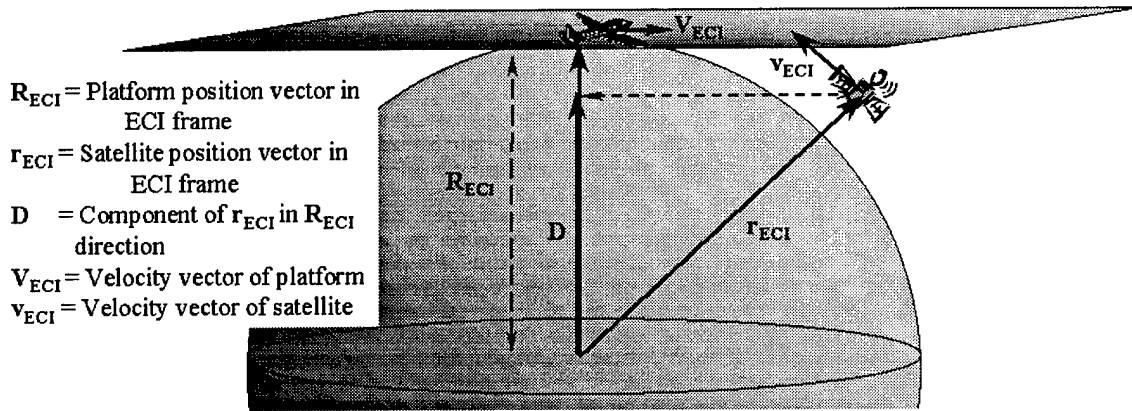
**Table 2.4. Possible Outcomes of Check to See if Satellite is Visible**

Case	Satellite	Satellite Ephemeris Path	Time To Rise	Visible To Platform?
1	In View	In View	N/A	YES
2	Not In View	Not In View	N/A	NO
3	Not In View	In View	< TUNR	YES
4	Not In View	In View	> TUNR	NO

To catch this minor error, we must make a check to ensure that the satellite is not going to rise above the artificial horizon of the platform, at least until after the next run of the preprocessor. If this is the case, then our satellite falls into case 4, otherwise, it must fall within case 3.

## 2.10 Checking Time-to-Rise of the Satellite

There are many ways to check the approximate time until the satellite rises above the artificial horizon. We will benefit from realizing that, for our application, we are only



**Figure 2.10. Vectors Used to Approximate Rise Time**

interested in the case where the satellite is very close to the artificial horizon, because the preprocessor, as mentioned before should be run every few minutes. Referring to Figure 2.10 above, we would like to find out how much time passes before the vector  $D$  has a greater magnitude than the position vector of the aircraft,  $R_{ECI}$ . To do this, we must find the rate of change of  $D$  with time:

$$\text{Time to Rise} = \frac{D - R}{\dot{D}} \quad (2.31)$$

Finding the derivative of  $D$  is a tricky process, but, because we are interested in only the time when the satellite is a few minutes away from the artificial horizon, we can find an approximation for the derivative of  $D$ :

$$\dot{D} \approx \bar{v}_{ECI} \cdot \hat{R} + \bar{r}_{ECI} \cdot \frac{\bar{V}_{ECI}}{|R_{\oplus} + h|} \quad (2.32)$$

In this equation,  $\mathbf{v}_{ECI}$  is the satellite velocity vector.  $\mathbf{R}$  is the unit vector in the direction of the platform position vector.  $\mathbf{V}_{ECI}$  is the platform velocity, while  $\mathbf{r}$  is the satellite position vector;  $R_{\oplus}$  is the radius of the Earth, and  $h$  is the height of the platform above the surface of the Earth. All of these vectors are in the ECI frame. We can easily obtain  $\mathbf{v}_{ECI}$  from our time propagator, and both the satellite and platform position vectors have already been calculated. This leaves only the velocity of the platform in the ECI frame to calculate. We must assume that we are given the velocity of the aircraft in the ECEF frame. Conversion to the ECI frame involves accounting for the angle,  $\theta_g$ , that currently separates the rotation of the ECEF frame with respect to the ECI frame, as well as the instantaneous rate,  $\omega$ , at which this angle is increasing. In short the velocity of the aircraft in the ECI frame will be:

$$\mathbf{V}_{ECI} = \begin{bmatrix} \cos \theta_g & -\sin \theta_g & 0 \\ \sin \theta_g & \cos \theta_g & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \vec{V}_{ECEF} + \begin{bmatrix} 0 \\ 0 \\ \frac{2\pi \text{ rad}}{86164.09054 \text{ sec}} \end{bmatrix} \times \vec{R}_{ECEF} \quad (2.33)$$

Once the rate of change of  $\mathbf{D}$  has been determined, the approximate rise time from equation 2.31 can be compared to the Time Until Next Run (TUNR).

**If TUNR > Time To Rise** → Include satellite in list given to Main Processor

**If TUNR < Time To Rise** → Throw out satellite because it is not visible

## 2.11 Preprocessor Methodology Conclusion

The goal of the ABLPA Preprocessor is to weed out all satellites that are not visible during a given time period. After running the software that models the algorithm described in this chapter, the results indicate that, given a worst-case scenario, at least

75% of the active satellites in the input file are discarded. This leaves 25% or less of the active satellites to be analyzed by the Main Processor. After the Preprocessor has finished, it sends the satellites that are in view to an output file, where they can be read by the Main Processor when the need arises. A more extensive analysis of the Preprocessor will be addressed in Chapter IV.

### **III. ABLPA Main Processor Methodology**

We have just finished discussing the ABLPA Preprocessor, which handles an input file of satellite Two-Line Element (TLE) sets, and forms an output file of all satellites in that list that are in view of the laser platform during a given time. This chapter will discuss the ABLPA Main Processor, which is designed to take this "shortened" list of satellites and perform real-time calculations to determine if any of these satellites will fall within the predicted arc of the laser during a given fire sequence. The first step in this calculation sequence is to determine the location of each satellite with respect to the platform.

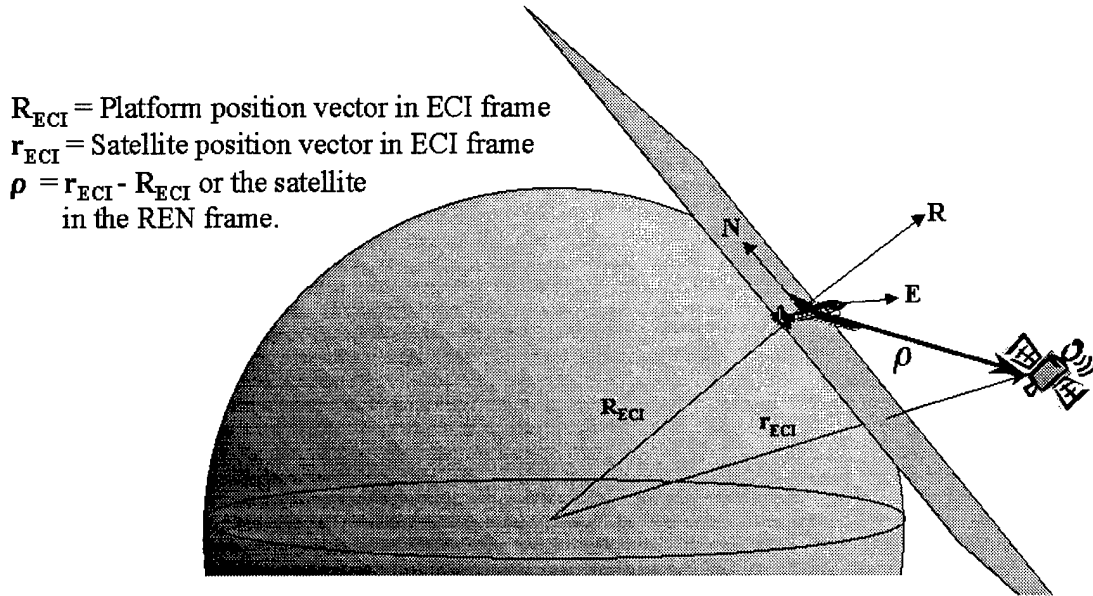
#### **3.1 Targeting the Satellite**

Up to now, we have looked at the positions of both the platform and the satellite as coordinates in the ECI frame. Now, however we wish to view the satellite as it appears with respect to the platform. This is done easily by switching to a new platform-centered coordinate frame.

##### **3.1.1 The REN frame**

This new platform-centered frame will be referred to as the Radial/East/North (REN) coordinate frame. The three right-handed axes will consist of the line from the platform to the north pole (the line tangent to the path as traveled across the spherical surface of the Earth), a line traveling due East, and a radial component "up" from the center of the Earth. In this coordinate frame, the platform position will be made the

center by subtracting its position from other bodies also referenced in the REN frame. A representation of this new coordinate frame is shown in Figure 3.1. Using this frame of reference, a new position vector,  $\rho$ , will refer to the position of the satellite with respect



**Figure 3.1. Derivation of  $\rho$  in ECI Frame With Respect to the REN Frame**

to the platform. The vector  $\rho$  can be derived from the two vectors  $R_{ECI}$ , the platform position in the ECI frame as referenced from the center of the Earth, and  $r_{ECI}$ , the satellite position vector in the same frame:

$$\bar{\rho}_{ECI} = \bar{r}_{ECI} - \bar{R}_{ECI} \quad (3.1)$$

where:

$$\bar{\rho}_{ECI} = \rho_x \hat{i} + \rho_y \hat{j} + \rho_z \hat{k} \quad (3.2)$$

we want to obtain  $\rho$  in the REN frame:

$$\bar{\rho}_{REN} = \rho_r \hat{R} + \rho_e \hat{E} + \rho_n \hat{N} \quad (3.3)$$

To find this vector in the REN frame, we must first find the conversion matrix that will take us from the ECI coordinate frame to the REN frame. This coordinate transformation matrix is simply:

$$\bar{\rho}_{REN} = \begin{bmatrix} \hat{\mathbf{R}} \cdot \hat{\mathbf{i}} & \hat{\mathbf{R}} \cdot \hat{\mathbf{j}} & \hat{\mathbf{R}} \cdot \hat{\mathbf{k}} \\ \hat{\mathbf{E}} \cdot \hat{\mathbf{i}} & \hat{\mathbf{E}} \cdot \hat{\mathbf{j}} & \hat{\mathbf{E}} \cdot \hat{\mathbf{k}} \\ \hat{\mathbf{N}} \cdot \hat{\mathbf{i}} & \hat{\mathbf{N}} \cdot \hat{\mathbf{j}} & \hat{\mathbf{N}} \cdot \hat{\mathbf{k}} \end{bmatrix} \cdot \begin{pmatrix} \rho_x \\ \rho_y \\ \rho_z \end{pmatrix}^T \quad (3.4)$$

While the  $\hat{\mathbf{i}}$ ,  $\hat{\mathbf{j}}$ , and  $\hat{\mathbf{k}}$  unit axis' of the ECI frame have been defined already, the  $\mathbf{R}$ ,  $\mathbf{E}$ , and  $\mathbf{N}$  unit axis' of the REN frame have not yet been rigorously defined. The  $\mathbf{R}$  unit axis direction can be seen to be the same direction as the position vector of the aircraft in the ECI frame (pointing up from the center of the Earth):

$$\hat{\mathbf{R}} = \frac{\bar{\mathbf{R}}_{ECI}}{|\bar{\mathbf{R}}_{ECI}|} \quad (3.5)$$

The  $\mathbf{E}$  unit direction can be derived from the angular motion of the Earth crossed with the aircraft position vector direction:

$$\hat{\mathbf{E}} = \frac{\bar{\omega}_{\oplus} \times \bar{\mathbf{R}}_{ECI}}{|\bar{\omega}_{\oplus} \times \bar{\mathbf{R}}_{ECI}|} \quad (3.6)$$

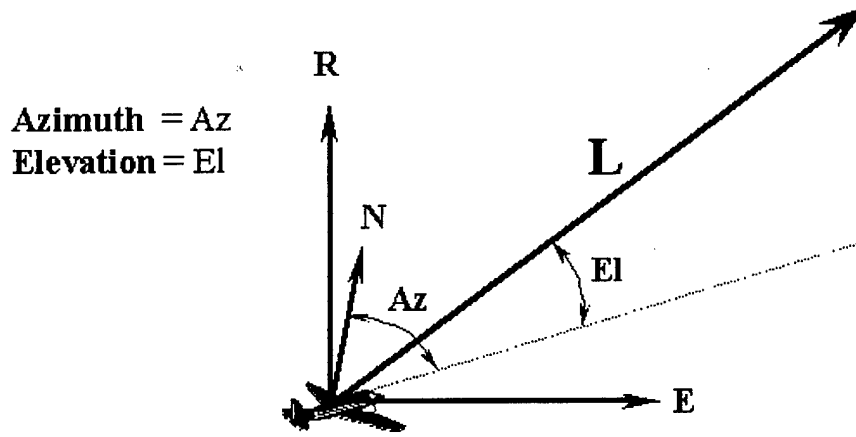
Similarly, because the REN coordinate frame is right-handed:

$$\hat{\mathbf{N}} = \hat{\mathbf{R}} \times \hat{\mathbf{E}} \quad (3.7)$$

Armed with this information,  $\rho$  can now be found in the REN frame.

### 3.1.2 Determining Laser Position in the REN Frame

Now that we have the position vector of the satellite with respect to the aircraft, it is a fairly easy matter to find the unit position vector of the laser. Assume that the laser Azimuth ( $Az$ ) and Elevation ( $El$ ) will be known.  $Az$  will be given in degrees east of north.  $El$  will be given in degrees above the platform artificial horizon.



**Figure 3.2. Laser Position in the REN Frame**

Looking at Figure 3.2, it is fairly straightforward to derive the unit position vector  $L$  in the REN frame.  $L$  describes the unit direction in which the laser is pointing in the REN frame, and has unit magnitude:

$$\hat{L} = \begin{bmatrix} \sin(El) \\ \cos(El)\sin(Az) \\ \cos(El)\cos(Az) \end{bmatrix} \cdot \begin{bmatrix} \hat{R} \\ \hat{E} \\ \hat{N} \end{bmatrix} \quad (3.8)$$

With these two vectors,  $\rho$  and  $L$ , we now have a way to compare the position of the satellite with the position of the laser turret, both of which are given in the REN frame.



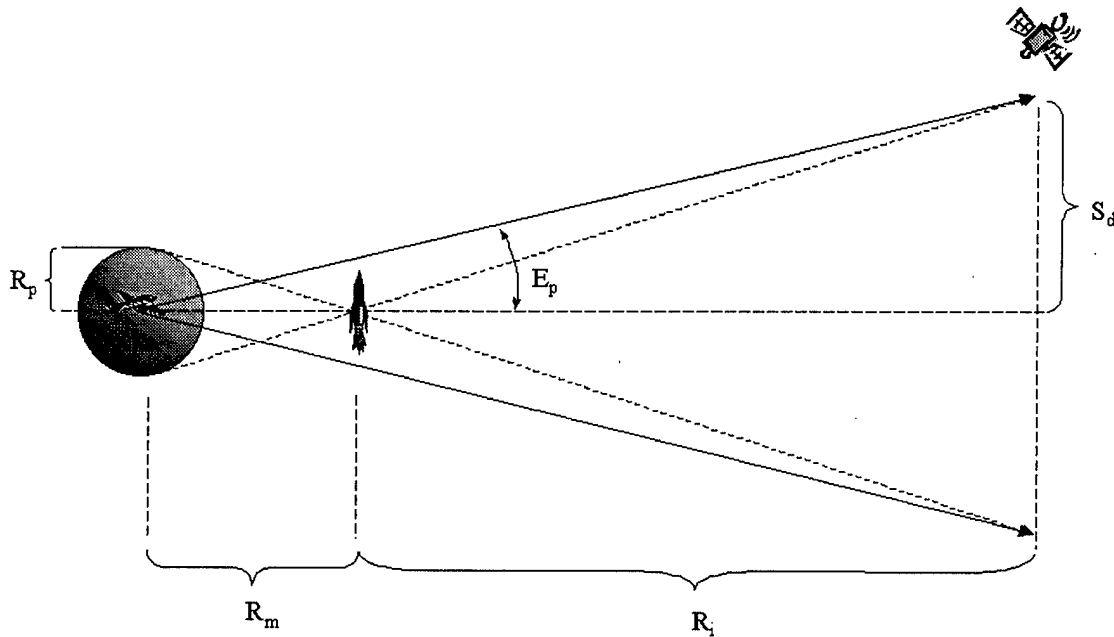
### **3.2 Position Error**

Up until this point, we have been addressing the position of the satellite and laser arc as points that are both fixed and known. However, applying this to an operational setting will quickly reveal that the exact position of both the laser and the satellite are only known within some degree of error. These errors introduce a growing uncertainty that must be modeled and accounted for if we are to reliably deconflict the path of the laser with the ephemeris of the satellite. There are a number of uncertainties that the reader may have noticed thus far. The first set of uncertainties involve the positions of the platform (or laser turret), satellite, and the missile. Our current level of technology allows us to establish and forecast positions in the ECI frame fairly accurately, but each position estimate or forecasted position estimate will still have an uncertainty. The second set of uncertainties concern the laser itself. Each of these errors will be addressed here.

#### **3.2.1 Platform Position Error**

The laser platform, in the case of the ABL program, is a Boeing 747-400 airframe equipped with a Honeywell GPS package. According to Honeywell, this GPS system will be accurate to within 10 meters. This being the case, we will have an instantaneous position error of only  $\pm 10$  m or .01 km. This error is fairly small. However, given the nature of the Predictive Avoidance mission, we have the need to forecast the plane's position into the future by approximately 20-30 seconds, in order to fully encompass the necessary laze time to destroy the missile. Recall that the firing solution must be derived in the second before the laser fires, so that the laser can fire for an uninterrupted amount of time. This amount of time is not expected to exceed 30 seconds. Therefore, we must

also know the path of the airplane in those 20-30 seconds. It is assumed by the author that the trajectory of the platform will be somehow actively controlled by an autopilot, so that any deviations from the planned course will be corrected in real-time. Given this type of active control system, the author assumes that the position error of the plane, working in conjunction with the Honeywell GPS, should not exceed  $\pm 50$  meters or 0.05 kilometers.



**Figure 3.3. Computing the Error Angle Contributed By Platform Position Error**

Consider the illustration given in Figure 3.3. The inputs that are currently known are the absolute range to the satellite, the range to the missile,  $R_m$ , the position error of the platform,  $R_p$ , and the approximate intermediate distance between the missile and the satellite,  $R_i$ . Notice that  $R_i$  can be approximated, by subtracting  $R_m$  from the absolute range to the satellite when the satellite is close to intersection with the laser. The goal is to find the effective error angle,  $E_p$ , contributed by the unknown location of the platform.

From the initial known parameters, the downrange spread distance,  $S_d$ , can be approximated by:

$$S_d = \frac{R_p \cdot R_i}{R_m} \quad (3.9)$$

Knowing  $S_d$  allows the computation of the effective error angle  $E_p$ :

$$E_p = \tan^{-1}\left(\frac{S_d}{R_m + R_i}\right) \quad (3.10)$$

It can be seen here that  $E_p$  will have a bigger impact on the overall error angle as the range to the satellite increases. As the satellite moves farther away from the platform, the error angle due to satellite position uncertainty will decrease, while  $E_p$  will remain constant. Thus,  $E_p$  will play an ever more prominent role as the range to the satellite increases.

### 3.2.2 Satellite Position Error

The satellite position error is the dominant position error in the error budget for almost all cases. The position of the satellite, as mentioned before, is a forecast derived from SGP4. In the period of 20-30 seconds, the error ellipsoid in which the satellite can be expected to reside will not change significantly. However, from the outset, the error ellipsoid will be big. Without conducting a thorough study of SGP4, there is no concrete way to establish an exact error, or rate of change of error. Furthermore, at this time we do not know how current the satellite input to SGP4 will be. If SGP4 is expected to propagate over a period of hours, the error will be significantly less than if propagation is conducted over a period of days or weeks. In the absence of accurate data, and realizing that the propagation is likely to be less than a day, it will be assumed that the position of

the satellite is established at  $\pm 10000$  meters or 10 km. Given 30 years of satellite tracking history with SGP4, this position error estimate appears to be a reasonable average. The satellite position error will also be a parameter in the software that can be changed as the customer, if deemed necessary. Assuming that the position error radius of the satellite,  $R_s$ , can be approximated, the error angle contributed by the satellite will be:

$$E_s = \tan^{-1}\left(\frac{R_s}{|\rho|}\right) \quad (3.11)$$

Where  $|\rho|$  represents the range to the satellite. Notice that this error angle will decrease as the range to the satellite increases.

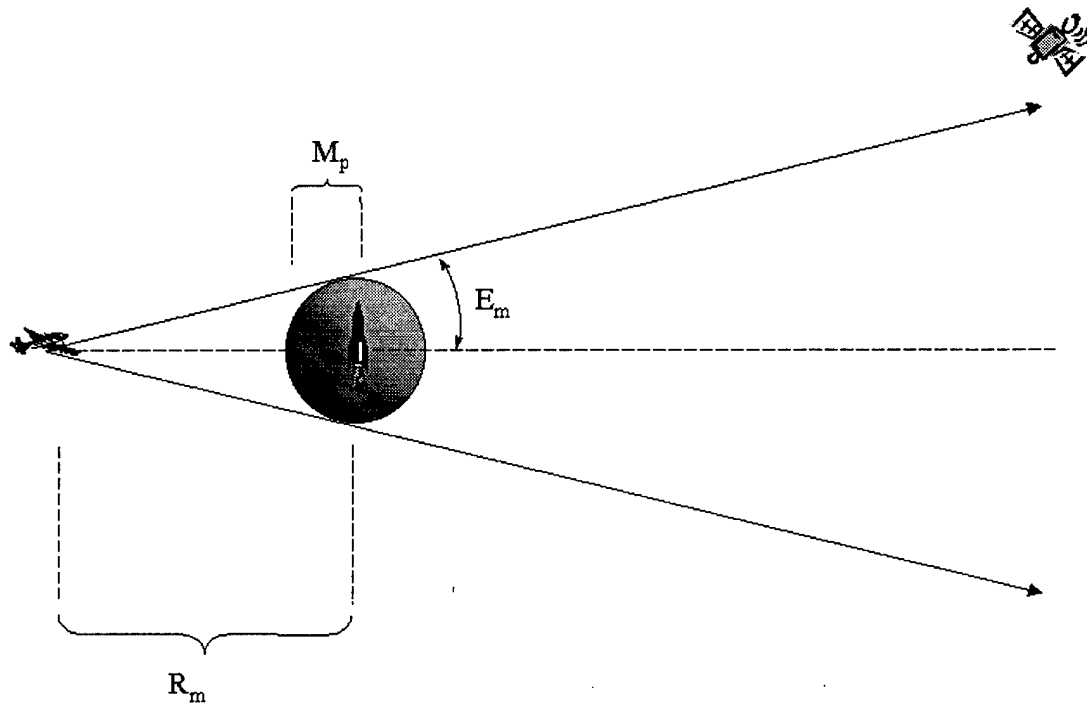
### 3.2.3 Missile Position Error

Up to this point, the missile's position has been thought of only as an extension from the laser turret. At the time of laze, the turret line of sight must be positioned on the missile such that there is no more than  $\pm 1$ -2 meters of error. If this is not so then the laser will never reach its target, and the ABL program in general is in serious jeopardy. The range to the missile is also assumed to be known within 1-2 meters. What we do not know is the behavior of the missile when forecasted over time. It has been assumed that the turret slew rates currently used to keep the missile locked will not change over our 30-second forecast span. Hence, with regard to the turret slew rate,  $S$ , we must assume that:

$$\begin{aligned} \dot{S}_{forecast} &= \dot{S}_{lock} & \text{and} & & (3.12) \\ \ddot{S}_{forecast} &= \ddot{S}_{lock} \end{aligned}$$

It is not necessarily reasonable to expect that this will be true. The ABL platform is intended to intercept a missile during its initial boost stage, where it is the most

predictable and vulnerable. If this is the case, then the position error of the missile is orders of magnitude less than the position error of the satellite, with a position error sphere of roughly  $\pm 50$  meters.



**Figure 3.4. Illustration of the Error Angle Introduced by Uncertain Missile Position**

Referring to Figure 3.4, if the radius of this missile position error sphere,  $M_p$ , and the range to the missile,  $R_m$ , are known, then the error angle introduced by the missile position uncertainty,  $E_m$ , will be:

$$E_m = \tan^{-1}\left(\frac{M_p}{R_m}\right) \quad (3.13)$$

Notice that the error angle will increase as the platform gets closer to the missile. Also notice that, if the missile behaves in such a way that significant discrepancies are introduced into its trajectory, then the position error can easily increase from 50 meters to

5 kilometers. There are two ways to account for this error in a forecast. The first method is to introduce a significantly larger missile position error into the forecast, which will inevitably cause a much bigger error angle in our forecast. This is not very desirable, as it increases our error angle as much as two full degrees, and gives the user a smaller space with which to work. The second method is to recompute in mid-laze, should the lazing parameters stray considerably from initial conditions. While this may seem at first to also be undesirable, it can be seen that this method tightens the error angle considerably, allowing a substantially reduced chance that a forecast will result in a satellite hit. If no satellites were previously forecast to be intersected, then rerunning the Main Processor in mid-laze presents only an extremely small chance of forecasting an intersection of a satellite. The alternative to recomputation is to incorporate a position compensation error of two or more kilometers into the position error of the missile, adding whole degrees to our error angle.

### **3.3 Laser Diffraction Errors**

Normally one would think of a laser beam as being fairly concentrated, without much diffraction error. This is essentially the case between the platform and the missile, where the range is only 200 km or so, and the beam is "focused" on the target. However, when dealing with the range to an orbiting satellite, with distances of six Earth radii (GEO altitude), beam diffraction has the potential to become a significant factor. There are two areas that we must consider when attempting to determine how a beam will spread out over long distances. The first area deals with intrinsic errors within the laser itself, and the second area covers optical diffraction and divergence.

### 3.3.1 Laser Intrinsic Spread Error

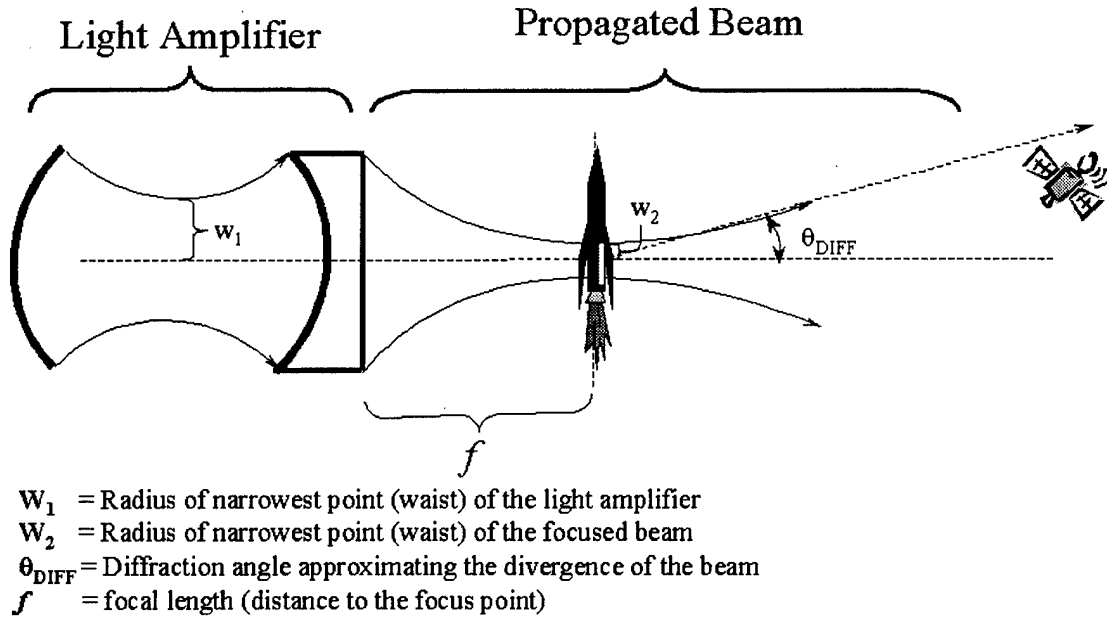
In theory, a laser should have no intrinsic spreading error, as every photon is at the same wavelength and has the same propagation direction. In practice, however, every laser has imperfections in its optics and propagation that cause the beam to diverge. These divergent patterns may be caused by very small imperfections in the alignment of the optics, transmission medium, or by a host of other factors. Unfortunately the intrinsic spread error of the ABL lasers have not been made available to us, and so the role it plays cannot be accurately discussed. However, we must assume that a laser designed to hit a 2 meter square target at 200 km is also designed to have a very small intrinsic error. Therefore we will assume this spread to be small enough to ignore, when compared with the significantly larger error angles introduced by position errors. It is mentioned here only for the purposes of pointing out that it has a potential for becoming a significant error source.

### 3.3.2 Optical Diffraction and Beam Divergence

The ABL High Energy Laser (HEL) is a “strongly focused” laser. That is, the optics are designed so that the beam will narrow from the aperture diameter of approximately 1.5 meters to the theoretical “diffraction-limited spot-size”, which is the smallest possible spot to which the laser can focus to. An illustration of this is shown in Figure 3.5. From this simple illustration it can be seen that the laser beam diverges in a parabolic path, but slowly conforms to a diffraction angle,  $\theta_{DIFF}$ . The angle,  $\theta_{DIFF}$  can be modeled by the equation:

$$\theta_{DIFF} = \frac{\lambda}{\pi w_2} \quad (3.14)$$

Where  $\lambda$  is the wavelength of the laser (in this case assumed to be 1.315  $\mu\text{m}$  for the HEL,



**Figure 3.5. Exaggerated Divergence of a Highly Focused Beam**

1.03  $\mu\text{m}$  for the TILL, or 1.06  $\mu\text{m}$  for the BILL), and  $w_2$  is the radius of the narrowest part of the focused beam.  $w_2$ , in turn can be found if  $w_1$  is known:

$$w_2 = \frac{f\lambda}{\pi w_1} \quad (3.15)$$

Where  $w_1$  represents the radius of the narrowest point, or “waist” of the radiation amplification chamber of the laser, and  $f$  is the focal length. Unfortunately, we are not privy to the specifications of the laser mechanism, so we need to find another way to estimate  $w_2$ . We can, for instance, assume that the designers of the ABL HEL system will wish to focus the laser as close to the theoretical limit as possible, approaching the diffraction-limited spot size:

$$\text{Spot Size} = f \tan\left(\frac{\lambda}{D}\right) \quad (3.16)$$



Setting  $w_2$  equal to the diffraction limited spot size should present a fairly close estimate to the actual value of  $w_2$ , and would yield the equation:

$$\theta_{DIFF} = \frac{\lambda}{\pi f \tan(\frac{\lambda}{D})} \quad (3.17)$$

In this way, we can approximate the beam dispersion by knowing only the focal length and aperture size of the laser.

### 3.4 Other Error Considerations

There are other errors that can pop up in our estimation of the probability to laze a satellite. The most notable error is the uncertainty that the satellite is traveling with constant speed and direction. Our time propagator assumes there are no forces acting on the satellite other than the pre-existing conditions and two-body gravitational effects. However, if the satellite undergoes station-keeping maneuvers, or is otherwise deflected from its course by some unforeseen event, the error can quickly become catastrophically worse over a surprisingly short period of time. Now the question must be asked as to whether or not we should choose to try to model these events and somehow account for them. This author would argue that the answer is no. To attempt to trap all possible errors, even those occurring beyond the  $3\sigma$  realm of possibility, would constrain the problem to such a degree as to make it unwieldy without adding much fidelity. We must assume that the satellite TLE files are only hours (at most) old. In this time, it is conceivable that a few station-keeping maneuvers have occurred on a few satellites. However, the chances that a given satellite has been maneuvered within a few hours **and** happens to cross the laser beam at the time it is fired are so infinitesimal as to preclude them from consideration. Such a chance certainly does not justify adding an additional

two degrees or so to every LEO satellite position error angle! Therefore, the possibility of station-keeping maneuvers is best left out of the project. Some of this error could, however be accounted for by increasing the position error of the satellite, if desired.

A second problem with the error analysis seen in the previous pages is that all position errors are modeled as a sphere. For the platform position error, this approximation may be accurate. However, for the forecasted missile and satellite positions, it is unlikely that this is the case. The missile's position, for example, is initially known to within a meter of our reference point, the platform. As time progresses within the forecast, the missile's position error will likely radiate from its known starting point as a elongated ellipsoid, rather than a sphere, as the acceleration is the most uncertain parameter, not necessarily direction. Furthermore, the satellite's position error is probably more accurately modeled as a highly eccentric ellipse, because its altitude is likely to be more established than its orbital progress. In future iterations of this project, it may be beneficial to model these position errors using a covariance matrix, rather than an absolute error angle.

### **3.5 Error Budget Consolidation**

Now that we have assessed the errors that exist within our error budget, we can begin to consolidate these errors into one error angle. You will notice that, of all the errors, Satellite Position Error seems to be the biggest in most cases. This is demonstrated in Table 3.1, which shows the magnitude of both the error angle as seen from the platform, and the position/displacement error as seen downrange at the satellite's distance from the platform, at both a LEO and GEO range. Knowing each of

the error angles computed above allows computation of the overall error angle as seen by the platform.

**Table 3.1. Error Budget for Predictive Avoidance Using the High Energy Laser (HEL) with a Wavelength of 1.315  $\mu\text{m}$**

<b>Error</b>	<b>½ Angle Error At LEO Alt Range = 500 km</b>	<b>½ Angle Error At GEO Alt Range = 36,000 km</b>	<b>Displacement Error At LEO Range=500 km</b>	<b>Displacement Error At GEO Range = 36,000 km</b>
<b>Airplane Position Error = <math>\pm 50</math> m</b>	0.01424 deg	0.01424 deg	$\pm 124$ m	$\pm 8950$ m
<b>Missile Position Error = <math>\pm 50</math> m</b>	0.01432 deg	0.01432 deg	$\pm 125$ m	$\pm 9000$ m
<b>Satellite Position Error = <math>\pm 10000</math> m</b>	1.14576 deg	0.01592 deg	$\pm 10,000$ m	$\pm 10,000$ m
<b>Laser Divergence</b>	0.00014 deg	0.00014 deg	$\pm 1$ m	$\pm 88$ m
<b>Laser Intrinsic Spread</b>	Unknown, Assumed Small	Unknown, Assumed Small	Unknown, Assumed Small	Unknown, Assumed Small
<b>Total Error = <math>\sqrt{a^2 + b^2 + c^2 + d^2}</math></b>	<b>1.14588 deg</b>	<b>0.023072 deg</b>	<b><math>\pm 10,002</math> m</b>	<b><math>\pm 16,159</math> m</b>

The values in Table 3.1 were computed using a focal length,  $f$ , of 200 km (the range to the missile) and the HEL wavelength. Estimates for the BILL and TILL lasers will be very similar, as their wavelengths are fairly close to the HEL wavelength, and Laser Divergence is a minimal contribution to the overall error angle. Each of the error sources is independent of the others, so the Total Error is simply found by taking the square root of the sum of the squares of the individual errors. Thus, the overall error angle,  $\alpha$ , can be computed by knowing the error angle contributed by the position errors of the satellite,

missile, and platform ( $E_s$ ,  $E_m$ , and  $E_p$  respectively) as well as the Laser Divergence angle,

$\theta_{DIFF}$ :

$$\alpha = \sqrt{E_s^2 + E_m^2 + E_p^2 + \theta_{DIFF}^2} \quad (3.18)$$

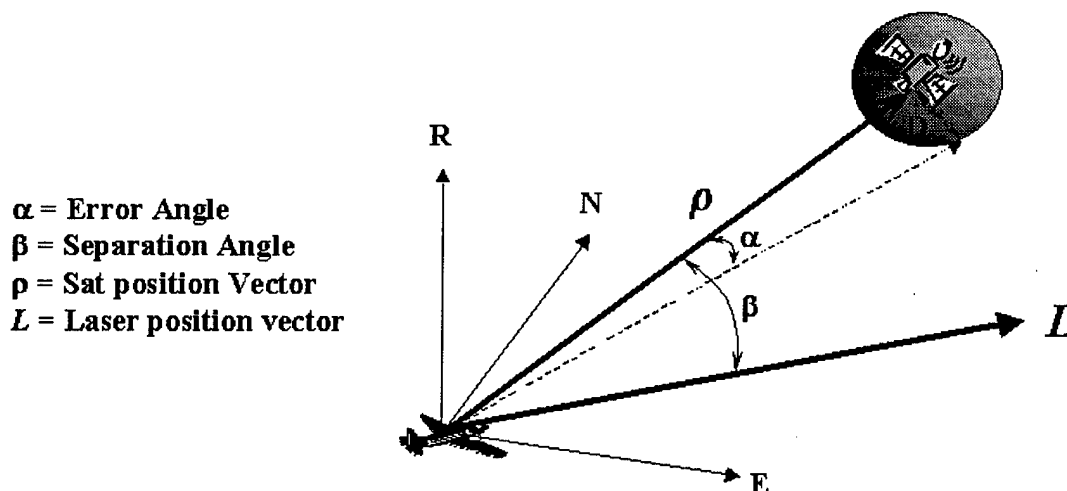
Notice that the error angle is largely dependent upon the range to the satellite in question. LEO orbiting satellites introduce a much larger error angle than the GEO satellites, because of their proximity to the platform. Also notice that the error angle is largely dominated by the position errors, which in every case establish at least 99.99% of the Total Error. This brings up an interesting short cut. As mentioned previously, the Main Processor is extremely constrained by processing time, because it must run in real time. Therefore we must ask whether or not the additional 0.01% error contributed by Laser Divergence is worth the extra processing necessary to calculate it. Almost certainly, it is not. Therefore the processor should be able to ignore this error and save a small amount of processing time without sacrificing much fidelity:

$$\alpha = \sqrt{E_s^2 + E_m^2 + E_p^2} \quad (3.19)$$

In the future, as position tracking for the ABL system becomes more refined, Laser Divergence error may play a bigger role, and thus have to be included in the error angle calculation. Until that time, the radius of our “position error cone” can be described as an angle,  $\alpha$ , which described in equation 3.19.

### 3.6 Finding the Current Separation Angle

To recap, first the satellite position vector,  $\rho$ , was computed in the REN frame, followed by the unit position vector of the laser,  $L$ , also in the REN frame. In the last section, the error angle,  $\alpha$ , was derived. Now, referring to Figure 3.6, everything is in place to find the angle,  $\beta$ , that separates the laser from satellite.



**Figure 3.6. Illustration of the Separation Angle**

Finding this angle is a fairly trivial matter:

$$\beta = \cos^{-1} \left( \frac{\bar{\rho} \cdot \hat{L}}{|\bar{\rho}|} \right) \quad (3.20)$$

However, forecasting the change of this angle with time is somewhat more involved.

### 3.7 Forecasting the Separation Angle

Previously, it was stated that the requirement exists to forecast the separation of the laser with a given satellite up to 30 seconds into the future. In order to attempt such a task, it is first necessary to know the rate of change of  $\beta$ , as well as its acceleration.

### 3.7.1 Finding the Rate of Change of the Separation Angle

The rate of change of  $\beta$  is simply its derivative. Recall from equation 3.20 that:

$$\beta = \cos^{-1}(u) \quad \text{where } u = \frac{\bar{\rho} \cdot \hat{L}}{|\bar{\rho}|} \quad (3.21)$$

Therefore, taking the derivative:

$$\dot{\beta} = \frac{-1}{\sqrt{1-u^2}} \frac{du}{dt} \quad (3.22)$$

where :

$$u = \frac{\bar{\rho} \cdot \hat{L}}{|\bar{\rho}|} \quad \text{and} \quad \frac{du}{dt} = \left( \frac{\bar{\rho} \cdot \dot{\hat{L}}}{|\bar{\rho}|} + \frac{\dot{\bar{\rho}} \cdot \hat{L}}{|\bar{\rho}|} - \frac{\bar{\rho} \cdot \hat{L}}{|\bar{\rho}|^2} \cdot \frac{\bar{\rho} \cdot \dot{\bar{\rho}}}{|\bar{\rho}|} \right)$$

From equation 3.22, the rate of change of  $\beta$  can now be computed, provided we can find the rate of change of the unit laser direction vector  $\hat{L}$ , and the rate of change of the satellite position vector,  $\rho$ . Recall from equation 3.8 that the initial laser position is given by its azimuth and elevation:

$$\hat{L} = \begin{bmatrix} \sin(El) \\ \cos(El) \sin(Az) \\ \cos(El) \cos(Az) \end{bmatrix} \cdot \begin{bmatrix} \hat{R} \\ \hat{E} \\ \hat{N} \end{bmatrix}$$

Assume also that the rate of change and accelerations of the Azimuth and Elevation,  $A\dot{z}$ ,  $A\ddot{z}$ ,  $E\dot{l}$ , and  $E\ddot{l}$  are also given. This is to be expected, as they will undoubtedly be made available from the electrical current controlling the laser tracking mechanism. This being the case, the rate of change of  $\hat{L}$  is easily found by taking the derivative:

$$\dot{\hat{L}} = \begin{bmatrix} \cos(El)E\dot{l} \\ \cos(El)\cos(Az)A\dot{z} - \sin(El)\sin(Az)E\dot{l} \\ -\cos(El)\cos(Az)A\dot{z} - \sin(El)\cos(Az)E\dot{l} \end{bmatrix} \cdot \begin{bmatrix} \hat{R} \\ \hat{E} \\ \hat{N} \end{bmatrix} \quad (3.22)$$

Now all that remains is to find the rate of change of  $\rho$ . Recall from equations 3.1 through 3.4 that:

$$\bar{\rho}_{ECI} = \bar{\mathbf{r}}_{ECI} - \bar{\mathbf{R}}_{ECI}$$

$$\bar{\rho}_{REN} = \begin{bmatrix} \hat{\mathbf{R}} \cdot \hat{\mathbf{i}} & \hat{\mathbf{R}} \cdot \hat{\mathbf{j}} & \hat{\mathbf{R}} \cdot \hat{\mathbf{k}} \\ \hat{\mathbf{E}} \cdot \hat{\mathbf{i}} & \hat{\mathbf{E}} \cdot \hat{\mathbf{j}} & \hat{\mathbf{E}} \cdot \hat{\mathbf{k}} \\ \hat{\mathbf{N}} \cdot \hat{\mathbf{i}} & \hat{\mathbf{N}} \cdot \hat{\mathbf{j}} & \hat{\mathbf{N}} \cdot \hat{\mathbf{k}} \end{bmatrix} \cdot \begin{pmatrix} \rho_x \\ \rho_y \\ \rho_z \end{pmatrix}^T$$

Where  $\hat{\mathbf{i}}$ ,  $\hat{\mathbf{j}}$ , and  $\hat{\mathbf{k}}$  are the X, Y and Z unit axis' of the ECI frame,  $\mathbf{r}_{ECI}$  is the position of the satellite,  $\mathbf{R}_{ECI}$  is the position of the aircraft, and  $\mathbf{R}$ ,  $\mathbf{E}$ , and  $\mathbf{N}$  are the unit directions of the REN frame. It stands to reason that the rate of change of  $\rho$  can be found in a similar manner:

$$\dot{\bar{\rho}}_{ECI} = \frac{d}{dt} \bar{\rho}_{ECI} = \frac{d}{dt} \bar{\mathbf{r}}_{ECI} - \frac{d}{dt} \bar{\mathbf{R}}_{ECI} \quad (3.23)$$

The reader may recall from Chapter 2 that the velocity of the platform has already been derived in the ECI coordinate frame in equation 2.33:

$$\frac{d}{dt} \bar{\mathbf{R}}_{ECI} = V_{ECI} = \begin{bmatrix} \cos \theta_g & -\sin \theta_g & 0 \\ \sin \theta_g & \cos \theta_g & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \bar{\mathbf{V}}_{ECEF} + \begin{bmatrix} 0 \\ 0 \\ \frac{2\pi \text{ rad}}{86164.09054 \text{ sec}} \end{bmatrix} \times \bar{\mathbf{R}}_{ECEF}$$

Furthermore, the velocity of the satellite in the ECI frame can be extracted directly from SGP4. Therefore all of the components necessary to find the rate of change of  $\rho$  in the ECI frame are available. All that remains is to find this rate of change with respect to the REN frame. This is done as is was previously, multiplying by the same conversion

matrix used to transfer the satellite position vector from the ECI frame into the REN frame:

$$\dot{\bar{\rho}}_{REN} = \begin{bmatrix} \hat{\mathbf{R}} \cdot \hat{\mathbf{i}} & \hat{\mathbf{R}} \cdot \hat{\mathbf{j}} & \hat{\mathbf{R}} \cdot \hat{\mathbf{k}} \\ \hat{\mathbf{E}} \cdot \hat{\mathbf{i}} & \hat{\mathbf{E}} \cdot \hat{\mathbf{j}} & \hat{\mathbf{E}} \cdot \hat{\mathbf{k}} \\ \hat{\mathbf{N}} \cdot \hat{\mathbf{i}} & \hat{\mathbf{N}} \cdot \hat{\mathbf{j}} & \hat{\mathbf{N}} \cdot \hat{\mathbf{k}} \end{bmatrix} \cdot \begin{pmatrix} \dot{\rho}_x \\ \dot{\rho}_y \\ \dot{\rho}_z \end{pmatrix}^T \quad (3.24)$$

Armed with the rate of change of  $\rho$  and  $L$ , the rate of change of the separation angle quickly follows from equation 3.22.

### 3.7.2 Finding the Acceleration of the Separation Angle

The only obstacle that remains in the quest to find an approximate forecast of the separation angle, is to find the rate of change of the rate of change, or the acceleration of the separation angle  $\beta$ . Recall from equation 3.22 that:

$$\dot{\beta} = \frac{-1}{\sqrt{1 - \left( \frac{\bar{\rho} \cdot \hat{L}}{|\bar{\rho}|} \right)^2}} \cdot \left( \frac{\bar{\rho} \cdot \dot{\hat{L}}}{|\bar{\rho}|} + \frac{\dot{\bar{\rho}} \cdot \hat{L}}{|\bar{\rho}|} - \frac{\bar{\rho} \cdot \hat{L}}{|\bar{\rho}|^2} \cdot \frac{\bar{\rho} \cdot \dot{\bar{\rho}}}{|\bar{\rho}|} \right)$$

Finding the acceleration, or the second derivative, of  $\beta$  requires taking the method used in the previous section one step further, and finding the derivative of  $\dot{\beta}$ . It may be easier to visualize the derivation of the acceleration by breaking the rate of change into two smaller functions:

$$\dot{\beta} = u \cdot v \quad (3.25)$$

where :

$$u = \frac{-1}{\sqrt{1 - \left( \frac{\bar{\rho} \cdot \hat{L}}{|\bar{\rho}|} \right)^2}} \quad \text{and} \quad v = \left( \frac{\bar{\rho} \cdot \dot{\hat{L}}}{|\bar{\rho}|} + \frac{\dot{\bar{\rho}} \cdot \hat{L}}{|\bar{\rho}|} - \frac{\bar{\rho} \cdot \hat{L}}{|\bar{\rho}|^2} \cdot \frac{\bar{\rho} \cdot \dot{\bar{\rho}}}{|\bar{\rho}|} \right)$$



From this it can be seen that:

$$\ddot{\beta} = u dv + v du \quad (3.26)$$

where :

$$u = \frac{-1}{\sqrt{1 - \left( \frac{\bar{\rho} \cdot \hat{L}}{|\bar{\rho}|} \right)^2}} \quad \text{and} \quad v = \left( \frac{\bar{\rho} \cdot \dot{\hat{L}}}{|\bar{\rho}|} + \frac{\dot{\bar{\rho}} \cdot \hat{L}}{|\bar{\rho}|} - \frac{\bar{\rho} \cdot \hat{L}}{|\bar{\rho}|^2} \cdot \frac{\bar{\rho} \cdot \dot{\bar{\rho}}}{|\bar{\rho}|} \right)$$

$$du = - \left( 1 - \left( \frac{\bar{\rho} \cdot \hat{L}}{|\bar{\rho}|} \right)^2 \right)^{-3/2} \cdot \left( \frac{\bar{\rho} \cdot \dot{\hat{L}}}{|\bar{\rho}|} + \frac{\dot{\bar{\rho}} \cdot \hat{L}}{|\bar{\rho}|} - \frac{\bar{\rho} \cdot \hat{L}}{|\bar{\rho}|^2} \cdot \frac{\bar{\rho} \cdot \dot{\bar{\rho}}}{|\bar{\rho}|} \right) \cdot \left( \frac{\bar{\rho} \cdot \hat{L}}{|\bar{\rho}|} \right) \quad \text{and}$$

$$dv = \left[ \frac{\bar{\rho} \cdot \ddot{\hat{L}}}{|\bar{\rho}|} + \frac{\dot{\bar{\rho}} \cdot \dot{\hat{L}}}{|\bar{\rho}|} - \frac{\bar{\rho} \cdot \dot{\hat{L}}}{|\bar{\rho}|^2} \cdot \frac{\bar{\rho} \cdot \dot{\bar{\rho}}}{|\bar{\rho}|} \right] +$$

$$\left[ \frac{\dot{\bar{\rho}} \cdot \dot{\hat{L}}}{|\bar{\rho}|} + \frac{\ddot{\bar{\rho}} \cdot \hat{L}}{|\bar{\rho}|} - \frac{\dot{\bar{\rho}} \cdot \hat{L}}{|\bar{\rho}|^2} \cdot \left( \frac{\bar{\rho} \cdot \dot{\bar{\rho}}}{|\bar{\rho}|} \right) \right] +$$

$$\left[ \left( \frac{\bar{\rho} \cdot \dot{\bar{\rho}}}{|\bar{\rho}|} \right) \cdot \left( \frac{2 \cdot (\dot{\bar{\rho}} \cdot \hat{L}) \cdot (\bar{\rho} \cdot \dot{\bar{\rho}})}{|\bar{\rho}|^4} - \left( \frac{\dot{\bar{\rho}} \cdot \hat{L} + \bar{\rho} \cdot \dot{\hat{L}}}{|\bar{\rho}|^2} \right) \right) \right] -$$

$$\left[ \left( \frac{\bar{\rho} \cdot \hat{L}}{|\bar{\rho}|^2} \right) \cdot \left( \frac{\ddot{\bar{\rho}} \cdot \dot{\bar{\rho}}}{|\bar{\rho}|} + \frac{\dot{\bar{\rho}} \cdot \ddot{\bar{\rho}}}{|\bar{\rho}|} - \frac{\bar{\rho} \cdot \ddot{\bar{\rho}}}{|\bar{\rho}|^2} \cdot \left( \frac{\bar{\rho} \cdot \dot{\bar{\rho}}}{|\bar{\rho}|} \right) \right) \right]$$

This somewhat longer equation for the acceleration of  $\beta$  brings with it two more terms that require further derivation. Just as finding  $\beta$  required finding  $\dot{\hat{L}}$  and  $\dot{\bar{\rho}}$ , so also the derivation of  $\ddot{\beta}$  introduces the variables  $\ddot{\hat{L}}$  and  $\ddot{\bar{\rho}}$ . Again, the derivation of the acceleration of  $L$  follows from the derivation of the rate of change of  $L$ . Recall that:

$$\dot{\hat{L}} = \begin{bmatrix} \cos(El)E\dot{l} \\ \cos(El)\cos(Az)A\dot{z} - \sin(El)\sin(Az)E\dot{l} \\ -\cos(El)\cos(Az)A\dot{z} - \sin(El)\cos(Az)E\dot{l} \end{bmatrix} \cdot \begin{bmatrix} \hat{\mathbf{R}} \\ \hat{\mathbf{E}} \\ \hat{\mathbf{N}} \end{bmatrix}$$

Therefore it follows that the acceleration of  $L$  will simply be the derivative with respect to time of the rate of change:

$$\ddot{\bar{L}} = \frac{d}{dt} \dot{\bar{L}} = \begin{bmatrix} \cos(El)E\ddot{l} - \sin(El)E\dot{l} \cdot E\dot{l} \\ \cos(El)\cos(Az)A\ddot{z} - A\dot{z}(\cos(El)\sin(Az)A\dot{z} + \sin(El)\cos(Az)E\dot{l}) - \\ \sin(El)\sin(Az)E\ddot{l} - E\dot{l}(\sin(El)\cos(Az)A\dot{z} + \cos(El)\sin(Az)E\dot{l}) \\ \cos(El)\sin(Az)A\ddot{z} + A\dot{z}(\cos(El)\cos(Az)A\dot{z} - \sin(El)\sin(Az)E\dot{l}) - \\ \sin(El)\cos(Az)E\ddot{l} - E\dot{l}(\cos(El)\cos(Az)E\dot{l} - \sin(El)\sin(Az)A\dot{z}) \end{bmatrix} \cdot \begin{bmatrix} \hat{\mathbf{R}} \\ \hat{\mathbf{E}} \\ \hat{\mathbf{N}} \end{bmatrix} \quad (3.27)$$

As stated previously, we have assumed that the acceleration of both the  $Az$  and the  $El$  are given. The only value left to derive is  $\ddot{\bar{\rho}}$ . Realizing that this can be found in the ECI frame and then converted to the REN frame in a similar manner as the rate of change derivation allows computation of the acceleration in the ECI frame:

$$\ddot{\bar{\rho}}_{ECI} = \frac{d^2}{dt^2} \bar{\rho}_{ECI} = \frac{d^2}{dt^2} \bar{\mathbf{r}}_{ECI} - \frac{d^2}{dt} \bar{\mathbf{R}}_{ECI} \quad (3.28)$$

Finding the acceleration,  $\ddot{\bar{\mathbf{r}}}$ , of the satellite at any given position is fairly straightforward in the ECI frame:

$$\frac{d^2}{dt^2} \bar{\mathbf{r}} = \frac{-\mu_{\oplus} \cdot \bar{\mathbf{r}}}{|\mathbf{r}|^3} \quad (3.29)$$

The gravitational constant for the Earth,  $\mu_{\oplus}$ , is roughly  $398601 \text{ km}^2/\text{sec}^3$ . Recall that the platform velocity in the ECI frame is:

$$\frac{d}{dt} \bar{\mathbf{R}}_{ECI} = \begin{bmatrix} \cos\theta_g & -\sin\theta_g & 0 \\ \sin\theta_g & \cos\theta_g & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \frac{d}{dt} \bar{\mathbf{R}}_{ECEF} + \bar{\omega}_{\oplus} \times \bar{\mathbf{R}}_{ECEF} \quad (3.30)$$

Where  $\bar{\omega}$  represents the angular rotation of the Earth as shown in equation 2.23. Because the platform is flying a fixed course, intentional acceleration due to course change should be zero, or very close to zero. Thus we are left with only the Coriolis and Centripetal accelerations in the acceleration derivative:

$$\frac{d^2}{dt^2} \bar{\mathbf{R}}_{ECI} = 2\bar{\omega} \times \begin{bmatrix} \cos \theta_g & -\sin \theta_g & 0 \\ \sin \theta_g & \cos \theta_g & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \frac{d}{dt} \bar{\mathbf{R}}_{ECEF} + \bar{\omega}_{\oplus} \times (\bar{\omega}_{\oplus} \times \bar{\mathbf{R}}_{ECEF}) \quad (3.31)$$

Both the first term in equation 3.31, the Coriolis acceleration, and the second term, the Centripetal acceleration, should be fairly small compared with the acceleration of the satellite, because the platform will not be “moving” nearly as fast as the satellite with respect to the ECI frame. Nonetheless, it has been decided to include the platform acceleration in the calculation of  $\ddot{\bar{\rho}}$ , even if only for theoretical completeness, as its inclusion does not significantly degrade software performance. After finding the acceleration of  $\bar{\rho}$  in the ECI frame, it can also be translated to the REN frame using the matrix employed in previous translations:

$$\ddot{\bar{\rho}}_{REN} = \begin{bmatrix} \hat{\mathbf{R}} \cdot \hat{\mathbf{i}} & \hat{\mathbf{R}} \cdot \hat{\mathbf{j}} & \hat{\mathbf{R}} \cdot \hat{\mathbf{k}} \\ \hat{\mathbf{E}} \cdot \hat{\mathbf{i}} & \hat{\mathbf{E}} \cdot \hat{\mathbf{j}} & \hat{\mathbf{E}} \cdot \hat{\mathbf{k}} \\ \hat{\mathbf{N}} \cdot \hat{\mathbf{i}} & \hat{\mathbf{N}} \cdot \hat{\mathbf{j}} & \hat{\mathbf{N}} \cdot \hat{\mathbf{k}} \end{bmatrix} \cdot \begin{pmatrix} \ddot{\rho}_x \\ \ddot{\rho}_y \\ \ddot{\rho}_z \end{pmatrix}^T \quad (3.32)$$

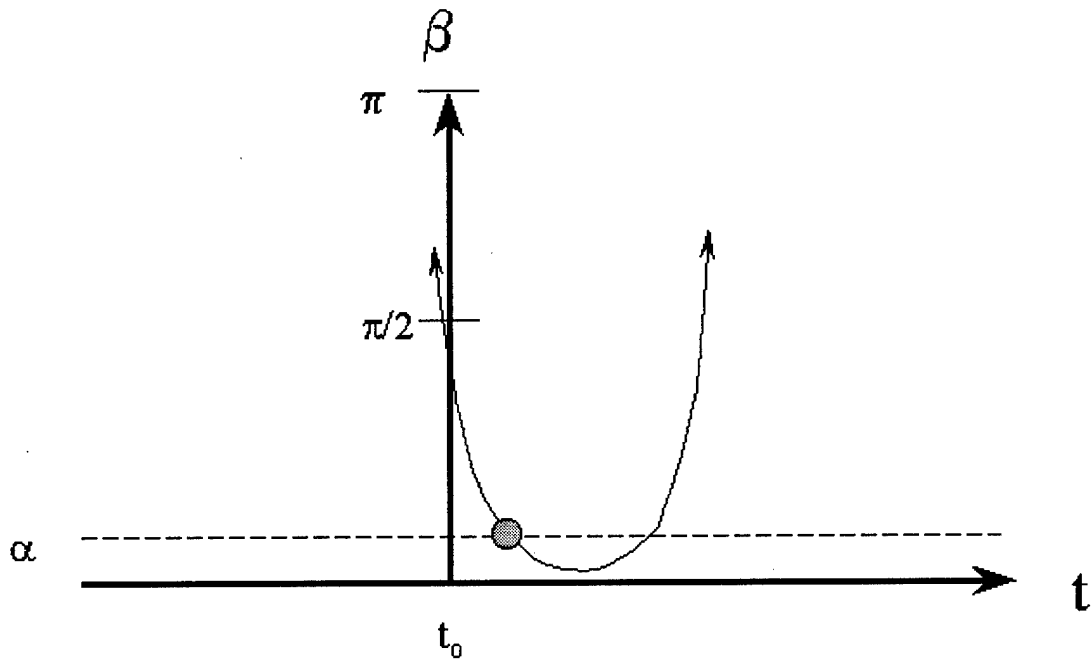
We now have all of the components necessary to build an initial forecast using the equations for  $\beta$ ,  $\dot{\beta}$ , and  $\ddot{\beta}$ , for the separation angle between the laser and the satellite at some given time in the future.

### 3.7.3 The Forecast Method

Now that  $\beta$ ,  $\dot{\beta}$ , and  $\ddot{\beta}$ , have been found from initial conditions, there are a number of ways to approximate  $\beta$  at a future time. The goal is to find the time (or times) that the laser will pass closer than the error angle to the exact predicted position of the satellite. Referring to Figure 3.6, the goal is to find any time in which the error angle,  $\alpha$ , is greater than or equal to the predicted value of  $\beta$ . Recognizing this, it can be seen that the forecast for  $\beta$  fits into a second-order Taylor series expansion:

$$\beta(t) = \alpha = \beta_o + \dot{\beta}\Delta t + \frac{1}{2}\ddot{\beta}\Delta t^2 \quad (3.33)$$

and solving for  $t$  will give us the times, if any, when intersects this region. This is illustrated in Figure 3.7. In actuality, the motion of the satellite with respect to the plane might better be described by a sine wave that varies in amplitude and frequency



**Figure 3.7. Illustration of a Satellite "Intersection".**

somewhere between zero and  $\pi$  radians as the days and weeks go by. However, for the short amount of time with which this project is concerned, we are only interested in the local minimums of this sine wave. These are the points of closest approach, and they can be approximated by using a second order parabolic function as described in equation 3.33. Later, the accuracy of this approximation will be addressed, but for now, it will be assumed that this approximation is accurate. To find the times at which this intersection will occur, the error angle can be brought inside the quadratic:

$$\beta(t) = 0 = (\beta_o - \alpha) + \dot{\beta}\Delta t + \frac{1}{2}\ddot{\beta}\Delta t^2 \quad (3.34)$$

And the quadratic equation can then be used to solve for  $\Delta t$ :

$$\Delta t = \frac{-\dot{\beta} \pm \sqrt{\dot{\beta}^2 - 2 \cdot \ddot{\beta}(\beta_o - \alpha)}}{\ddot{\beta}} \quad (3.35)$$

There will always be two solutions for  $\Delta t$ . However, in many cases, these solutions will be imaginary, because the terms inside the square root are negative. This would indicate that the separation angle never exactly equals the error angle, and thus never crosses it.

A second case might consist of two negative values for  $\Delta t$ , which would imply that the laser position vector has already passed through the satellite error cone, and is now headed away. The third case is that there is one positive root and one negative, which implies that the laser position vector is currently in the laser cone. This will be the case when  $\beta < \alpha$ , and can be recognized before the quadratic roots are ever found. The fourth case, in which  $\Delta t$  has two positive values, is the case in which an intersection is forecast to occur. These four cases are summarized in Table 3.2.

**Table 3.2. The Meanings of the Quadratic Roots for  $\Delta t$**

Quadratic Roots	Cause	Intersection?
Two Imaginary Roots	Laser never close enough to intersect the satellite	No
Two Positive Roots	Laser pos vector is moving away from satellite	No
One Positive, One Negative	Laser pos vector is currently intersecting the satellite	Yes
Two Negative Roots	Laser pos vector is moving towards satellite	Possibly

The first three cases are fairly straightforward. Either an intersection will occur or it will not. The fourth case however, demands a bit more attention. As an input to the algorithm, the Time of Laze,  $T_L$ , describing the expected duration of the laze is an important part of determining whether an intersection will occur. If the intersection is forecast to occur outside of the lazing window:

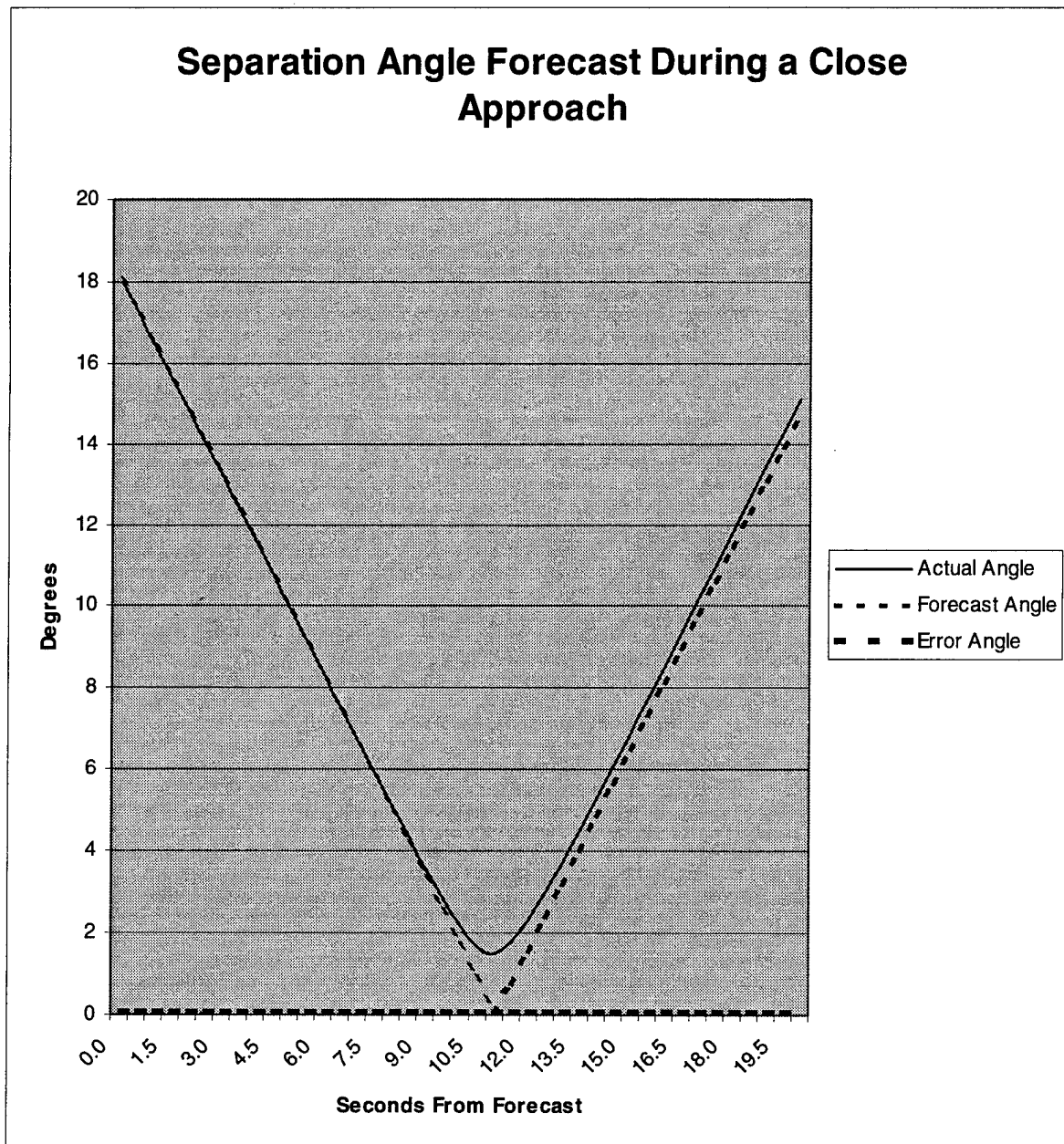
$$T_L \leq \Delta T_1 \quad (3.36)$$

where  $\Delta T_1$  represents the closest time to intersect, then, in fact, an intersection is not forecast to occur. If this is not the case, then a forecasted intersection has occurred.

### **3.7.4 Accuracy of the Forecast Method**

As mentioned previously, the method used to forecast the separation angle does not model the separation of the laser and satellite position vectors accurately in all situations. An example of an inaccuracy in the Forecast Method is illustrated in Figure 3.8. This figure shows the actual separation angles encountered in “close approach” of a satellite with the laser position vector. The “true angle” was compiled by actually interpolating the platform, laser, and satellite forward slowly in time, and plotting the

separation angle,  $\beta$  at each point in time. The reason this is established as the “actual separation angle” is that the positions of each of the players is fairly well known at a given time, and therefore  $\beta$  is equally definite. The unknown is how well our function,

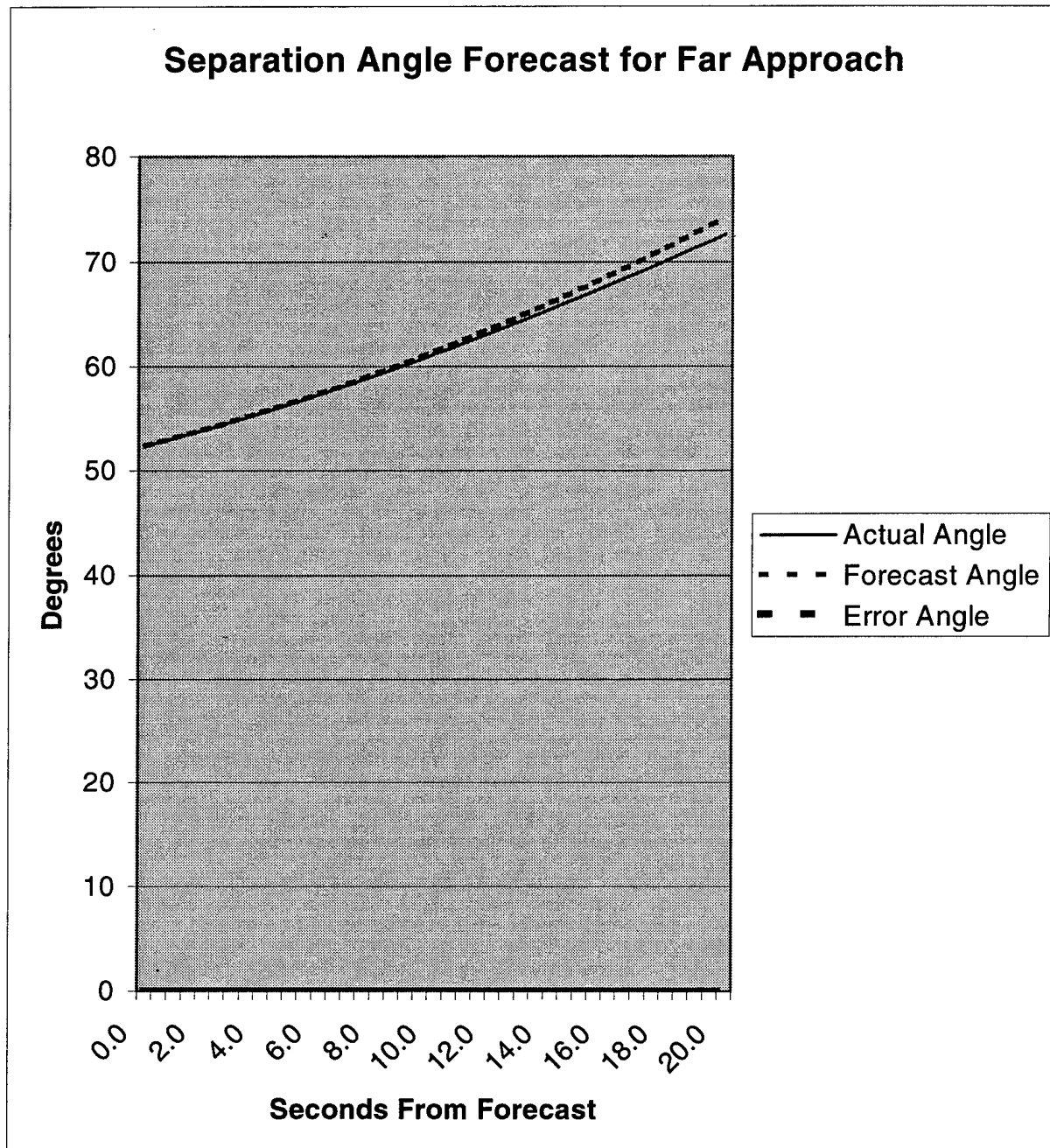


**Figure 3.8. Comparing the Actual Separation Angle Encountered in a Close Approach With the Forecasted Separation Angle**

that uses  $\dot{\beta}$  and  $\ddot{\beta}$  to anticipate its behavior, maps to the known behavior of  $\beta$ . In the case of Figure 3.8, the error angle was small, only about 0.075 degrees. Notice that at the time  $t=0$  seconds into the forecast, the forecast angle and the actual separation angle correspond almost perfectly. In fact the forecast angle matches the actual angle almost exactly, until 5 seconds before the closest approach, at which time it diverges ever more rapidly. Unfortunately, this is the period of time with which we are most concerned! The reasons for this divergence are interrelated. First, we are not using an approximation function that does not match exactly with the behavior of the actual angle. Second, during the time of closest approach, the initial conditions no longer predict the separation angle well. At this closest approach, the acceleration of the separation angle increases dramatically, due in large part to the proximity of the two vectors as they pass each other. This dramatic acceleration is seen in the rounded vertex of the actual separation angle. However, with the forecast, the acceleration remains minimal, pushing the forecast to actually intersect with the satellite at zero degrees of separation. This is, of course, a virtual statistical impossibility, and should raise considerable suspicion as to the veracity of the forecast. In fact it is to be expected that the forecast angle will deviate from the actual separation angle in *every* forecast, as the initial conditions will eventually no longer match actual conditions to the precision we require. For example, consider the 20-second forecast shown in Figure 3.9. This forecast deals with a satellite position vector that never gets closer than 40 degrees from the laser position vector. In this plot, it can be seen that the forecast occurred at a time when the two vectors were actually separating from each other. Despite their distance from each other, still the forecast deviates over time, as expected. So what does it profit to use this forecasting method?



The benefit of using this method to pinpoint intersections is that, in almost every case, the forecast will be conservative in its estimation of an intersection, and will locate the closest approach time to within 2 seconds of the actual closest approach time.



**Figure 3.9.** Illustration of Forecasted Angle Deviation From Actual Angle in a "Far Away" Satellite

Furthermore, the forecast will eliminate all but the most closely approaching satellite vectors. Of course the extent to which other satellites are eliminated by the forecast depends heavily upon the initial conditions. For example, having a laser turret slew rate of 10 degrees per second is likely to produce quite a few more "close-approach" satellites, like the one illustrated in Figure 3.8, than would a normal operational turret slew rate of, say, 1.5 degrees per second or less. Given the operational conditions, and the turret slew rates necessary to track a missile moving a roughly 3 kilometers a second at 100 kilometer range, it would be reasonable to expect that the turret would normally track at a rate of one to two degrees per second. At this rate, it is also reasonable to expect that the forecast method will eliminate at least 90% of the satellites in the list fed to it. This leaves us with, at most, 10% of the satellites given to the Main Processor that must be evaluated more closely.

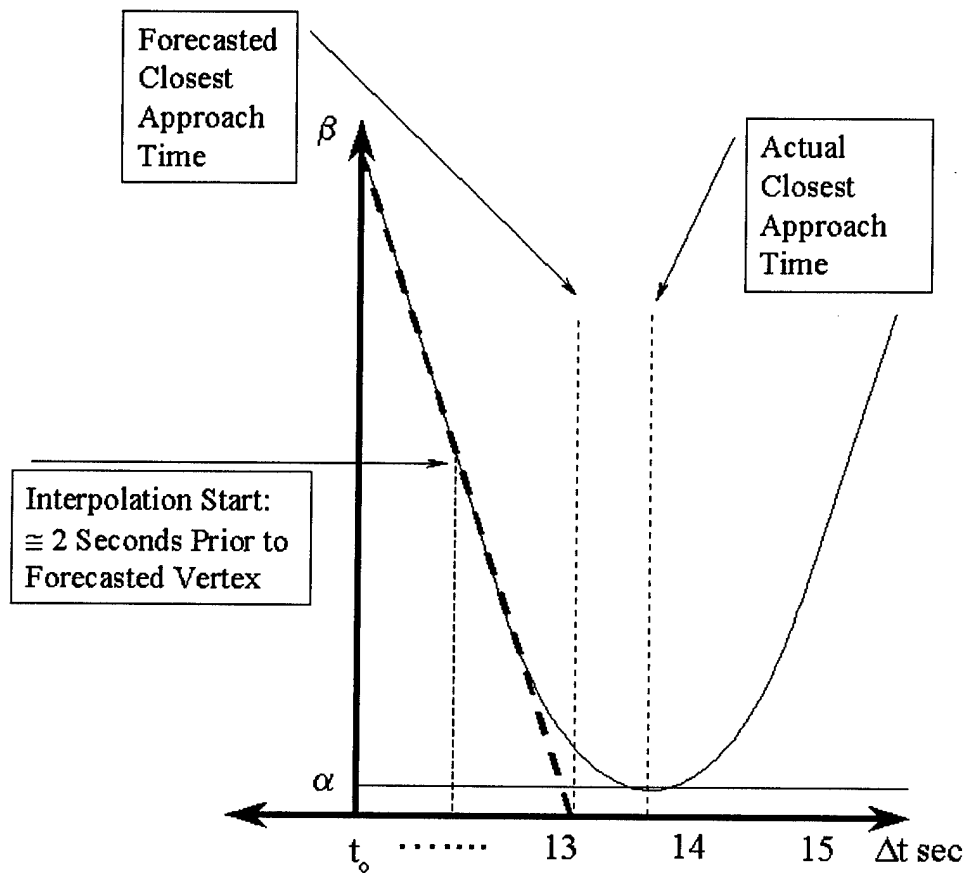
### **3.8 Interpolation to Correct the Forecast**

Now we have employed two filters to narrow the list of at-risk satellites. The first, the ABLPA Preprocessor, can be expected to eliminate roughly 75 of every 100 satellites in an operational environment, depending, to some degree, upon the location of the theater employment. If we start with the assumption that there are 1000 active satellites, this now leaves 250 to evaluate. The second filter, the Forecast Filter, can be expected to strain 90% of the remaining 250 and leave us with, at most, 25 at-risk satellites. With so few satellites remaining, it now becomes feasible to use straight interpolation of the separation angle for these few satellites over a given period of time. Fortunately, the Forecast Filter also gives a good approximation for the time at which the vertex, or the closest approach point of the satellite position vector to the laser position

vector, occurs. Knowing this, an interpolation can be set up using an Interpolation Time Buffer and Interpolation Step Size as the parameters controlling the interpolation.

### 3.8.1 The Interpolation Time Buffer

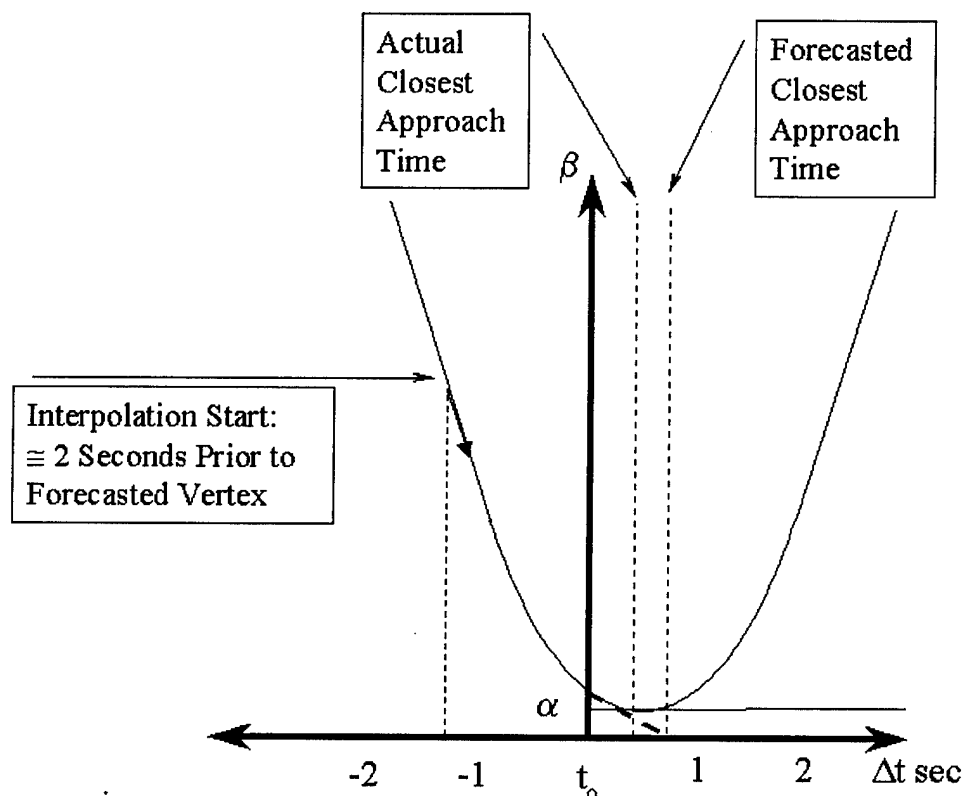
Starting with the vertex, a time increment can be specified as a certain interval before the forecasted vertex at which to begin interpolation. The vertex arrived at during the forecast does not necessarily occur at the time of closest approach. For example, Figure 3.10 illustrates the situation that will occur in almost all cases.



**Figure 3.10. Typical Early Vertex Forecast with a Two Second Interpolation Buffer**

In the typical forecast, when the satellite and laser approach each other within a few degrees, the actual separation angle will look fairly linear until a few seconds before the

closest approach, much like a triangle with a rounded bottom corner. This is seen clearly in the example given in Figure 3.8. With this type of function curve, typically the forecast vertex will occur slightly before the actual vertex in time. This is seen in Figure 3.10. So why not simply begin interpolation at the forecast vertex and move forward until the actual vertex is encountered? There are cases where the forecast vertex falls after the actual vertex in time. The first case can occur if our forecast is done near the vertex, before crossing the error angle. This is illustrated in Figure 3.11. Here, the



**Figure 3.11. Special Case Forecast Where Forecast Vertex Falls After Actual Vertex in Time**

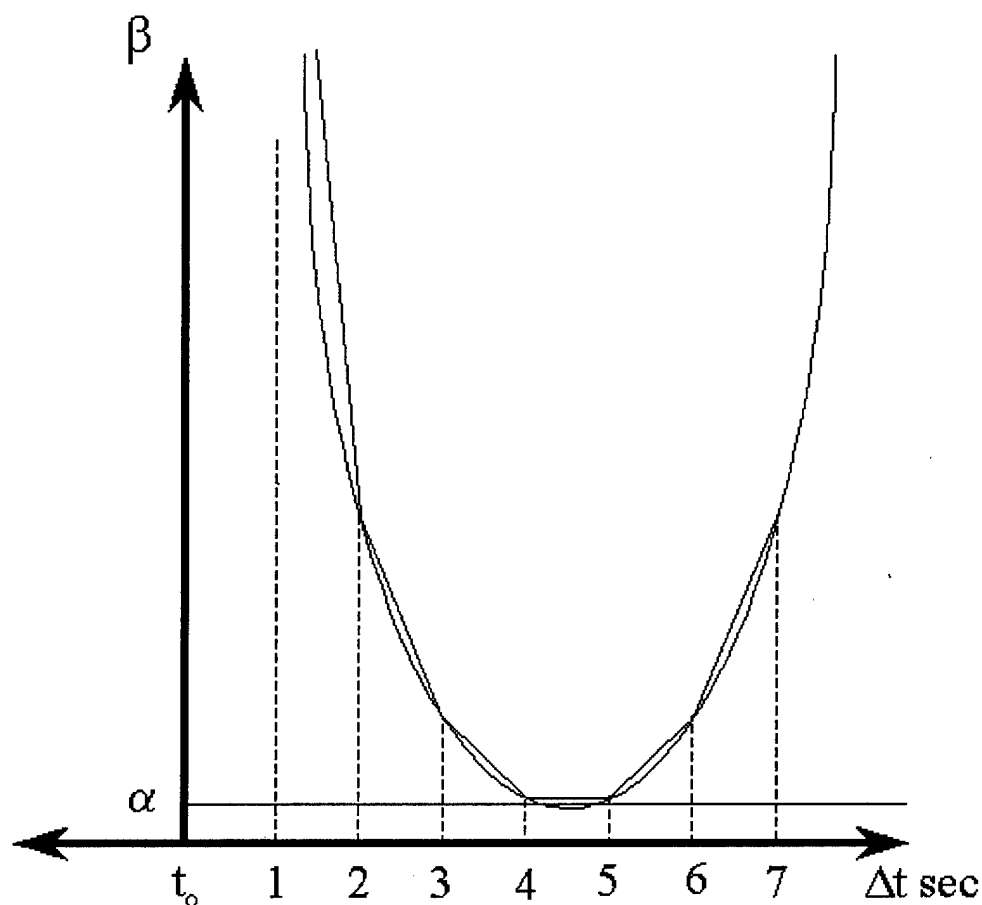
forecast occurs at a critical time near the vertex, but before the separation angle is below the error angle, so an initial check will not show the intersection, and the forecast will lie

beyond the actual vertex. It is for cases like this that a time buffer should be inserted between the forecast vertex and the interpolation start time, to ensure that the interpolation covers a big enough time block to include the actual vertex. Another case that may allow the forecast vertex to occur before the actual one, is the case when the error angle is much bigger. The example in Figure 3.10 is using an error angle that, although it is not labeled, would be only about 0.1 degrees. According to earlier analysis, however, it is anticipated that error angles of up to 1.15 degrees may be anticipated for LEO satellites. This will raise the possibility of a separation function that intersects the error angle at a higher degree. Such an event would cause the forecast to drift to the right of the actual intersection. A third case could occur where both case 1 and 2 happen, pushing the forecast even further ahead in time, although this is only a very remote possibility. The best time buffer to use is best left to the analyst who is using the algorithm. However, in this project, the time buffer was kept at two seconds. This buffer size allowed the algorithm to handle every variation and case that was run, without exception. This is not to say, however, that a case does not exist in which two seconds is insufficient. For this reason, the time interpolation buffer length is an input to the software, as opposed to a constant.

### **3.8.2 Interpolation Step Size**

Another time increment, the interpolation step size, must also be specified. The step size refers to the amount of time that transpires between samplings of the separation angle. It is the step size that determines the precision of the interpolation. If the step size is too big, then the true vertex may never be reached with enough precision to determine whether or not it crosses the error angle boundary. This is the case in Figure 3.12. In this

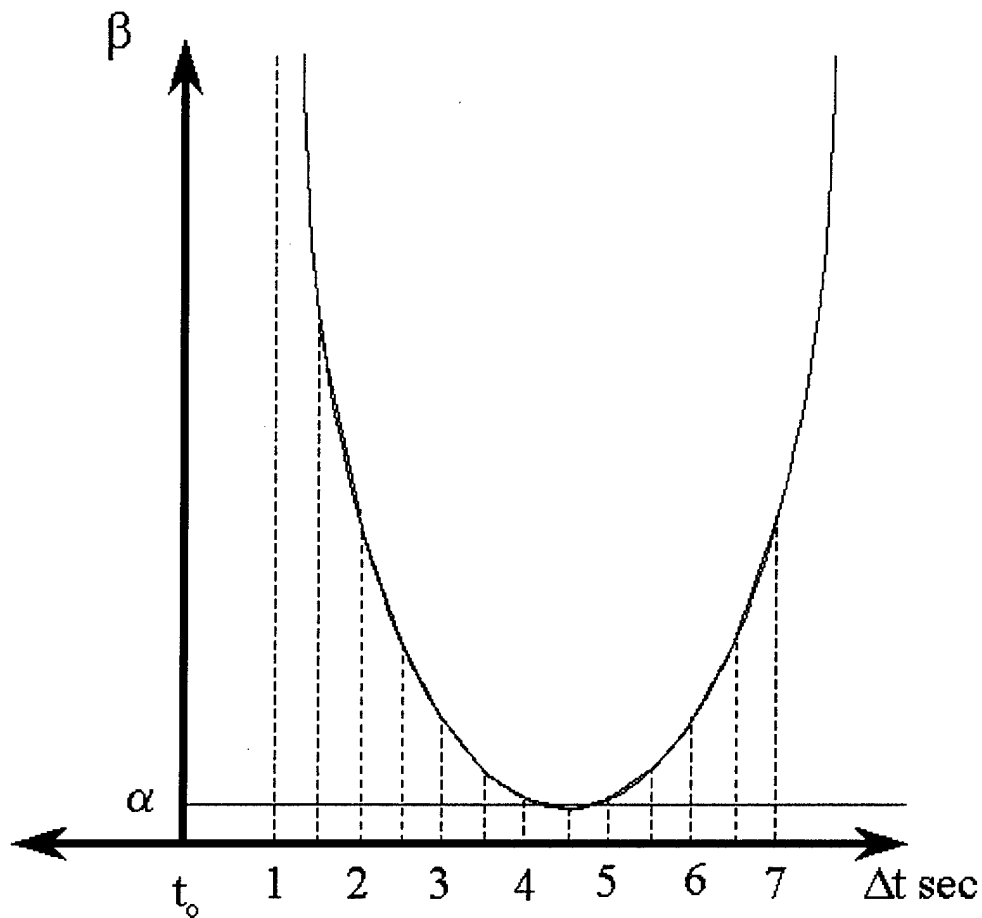
figure, the step size is one second. It can be seen that, although the separation angle crosses the error angle, The interpolated parabola never crosses the critical angle, and thus the intersection is never registered.



**Figure 3.12. Interpolation With a Step Size that is Too Large**

The obvious cure for this problem is to shorten the step size. However, this must be done with care. Each step represents a complete analysis of the extent of Earth's rotation, platform movement, laser turret tracking and a call to SGP4 (or another ephemeris propagator) to find the separation angle. If the step size is reduced from one second to 0.0001 seconds, then we have increased the number of analyses by a factor of ten

thousand (per satellite)! Our dilemma then, is to find the step size with enough precision to satisfy requirements. Figure 3.13 presents the same parabola as in Figure 3.12, but with exactly one-half the step size. Fortunately, the precision increases significantly as the step size decreases.



**Figure 3.13. Decreasing Step Size Increases Precision**

This project was tested with a step size of 0.1 seconds. This increment caught every satellite correctly for every operational scenario tested. Again, this is not to say that 0.1 seconds is necessarily the optimum increment, only that it worked flawlessly during testing.

### **3.9 Main Processor Methodology Conclusion**

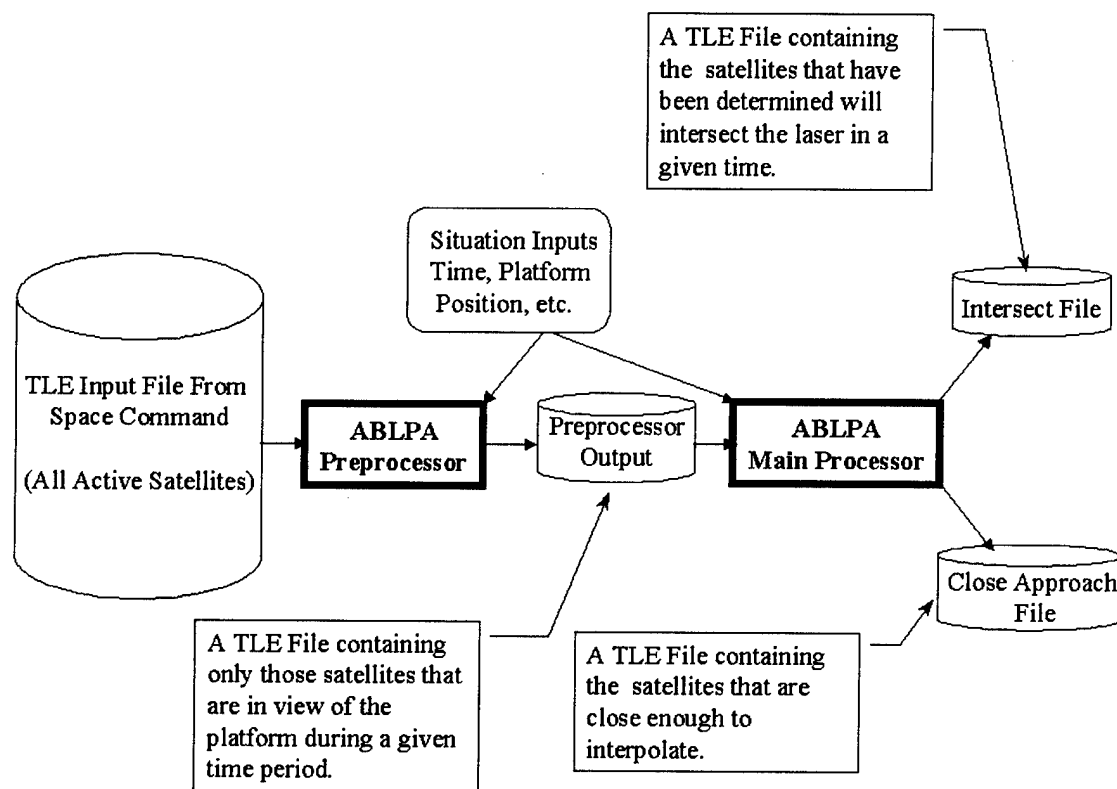
With this summary the theory and methodology used in this Predictive Avoidance project is concluded. While the previous pages focus on the algorithm that was developed for this project, *Chapter IV – Software Development* will attempt to describe the actual application of this algorithm in a software development project. Analysis of the algorithms presented in these last two chapters shall be fully addressed in *Chapter V - Analysis and Conclusions*.



## IV. Software Development

The software by which the algorithms discussed previously can be used practically is discussed briefly in this chapter. For a more comprehensive discussion of the software implementation, the reader may wish to refer to *Appendices A-F*. The Predictive Avoidance software has been written with the intent that it may be used within the fire control system of the Anti-Ballistic missile Laser (ABL) currently being developed by the Boeing Corporation. The software has been written in a modular format, and has been broken into logically functional modules that have been designed to work both together and independently, if needed. As mentioned in the Introduction, this software package has been designed with three conflicting (but important) objectives. The first objective is to make the software readily understandable to a person who wishes to study it in the future. The package is designed with an agreement by Boeing that it will be studied and at least partially incorporated directly into the BC/FC of the ABL platform. Therefore, to ensure a smooth incorporation into the ABL project, the most important consideration is that an engineer can survey the code and easily understand its content and purpose. The second goal is that the software be fast. It is estimated that the prediction avoidance software should not need more than 0.5 seconds to fully process a mission. Therefore algorithms should be designed to minimize processing time. The third major goal for the PA software is that it should be modular. Of these three goals for the software, understandability is the by far the most important. There are many cases within the software in which a fluid, slow, understandable implementation has been used instead of a speedier vague implementation. This is done with the understanding that the

software will be reviewed at a later time, when any “slow” algorithms may be supplanted with the software engineer’s choice of implementations. The standard flow of the software is shown in Figure 4.1.



**Figure 4.1. The Predictive Avoidance Software Flow**

#### 4.1 Modularity and Testing

As mentioned previously, modularity is a goal for this software package. Although the software package has been broken into two separate entities, namely the Preprocessor and the Main Processor, the modules that compose this project have been grouped together into twelve smaller libraries, by their logical functionality. The reason for this breakdown is simple. The project as a whole is difficult to test as a whole.

Breaking down the project into twelve testable mini-projects allows independent testing of a portion of the software while divorcing it from the whole. The benefits to this are intuitively obvious. Understandability and testability are increased, however, creation of this type of infrastructure in the project has also required that the code be that much larger, with more physical modules and interfaces. Each library is distinguished by having its own stand-alone GUI that calls the module(s) being used in that library. As implied before, there will be twelve software libraries total. Two of these will be the final ABLPA Preprocessor and Main Processor. The other ten will be composed of lower and lower levels of subordinate modules; five testing modules in the Preprocessor, and five more introduced in the Main Processor. The overall structure of the software is addressed with more depth in *Appendix A – Software Structure*.

#### **4.2 The Calculation Modules**

The modules that house the meat of the algorithm and the calculation intensive software are written in the C++ language. The compiler used is Borland C++ Builder 3 (Standard, C++ Version 5). It is likely that the code for these modules can be recompiled with minimal effort using another similar C++ compiler. These modules are written for easy adaptability to other environments. All of the code for the calculation modules will be included and discussed, as its explanation and operation is one of the important products that this thesis delivers. *Appendix B - The ABLPA Preprocessor Software* will discuss all modules developed for the ABLPA Preprocessor. *Appendix C - The ABLPA Main Processor Software* will discuss modules developed for the Main Processor that have not already been covered in *Appendix B*.

### **4.3 The Test Modules for Each Software Library**

There are other modules that have been developed exclusively for the task of calling and interfacing with the Preprocessor the Main Processor, and each of the other ten libraries. They are strictly “front-end” interfaces for the calculation modules that can be used to run and test the algorithms that have been developed within the calculation modules. These test modules have been kept as “simple” as possible, while still maintaining ease-of-use, to avoid the necessity of “testing” the test modules. They consist of a graphical interface that collects input to the module(s) being tested, and the function call to those modules. The graphical nature of the test modules utilizes quite a bit of compiler-specific organization and terminology to allow easy development of these modules. Therefore, any change or recompilation of these graphical interface modules will require the user to have a copy of Borland C++ Builder 3 (Standard). This compiler will also be required if the user of this software wishes to run the fully compiled software packages that have included within them the GUI interfaces. The Test Modules and their software code is described further in Appendix E.

### **4.4 The Environment**

These modules have all been developed using a standard desktop IBM compatible computer, using a 200 MHz Intel Pentium processor. The modules are compiled to run in the Microsoft Windows 95 or 98 operating system environment. Operation in other environments has not been tested and is not guaranteed, especially the graphical test modules.

## 4.5 Sample Interfaces

As mentioned previously, the product, C++ Builder 3, made by Borland was used to craft a graphical front-end to each application designed. One GUI has been created for each functional task (or library) needed in the Preprocessor and Main Processor. Each of these modules, where practical, comes with a graphical front-end interface to allow testing of individual components. Both the Preprocessor and the Main Processor have

Aircraft Information		Greenwich Time		Greenwich Meridian Reference	
Aircraft Latitude Degrees	10 N	Calculation Year	1998	Reference Hour	4
Aircraft Latitude Minutes	0	Calculation Month (1-12)	8	Reference Minute	40
Aircraft Latitude Seconds	0	Calculation Day	14	Reference Second	1.299
Aircraft Longitude Degrees	55	Calculation Hour	3	Modified Julian Date	11029.0
Aircraft Longitude Minutes	0	Calculation Minute	58	Seconds To Next Run	60
Aircraft Longitude Seconds	0	Calculation Second	2.0		
Aircraft Altitude	129			True G (deg)	0
Aircraft Xdot (ECEF) km/h	0	Input/Output Files			
Aircraft Ydot (ECEF) km/h	300	Input TLE File to Preprocessor	PAData/TLEFile4.txt		
Aircraft Zdot (ECEF) km/h	0	Output TLE File to Preprocessor	PAData/PreprocessorOutput.txt		
Satellites In Range (SSC #)	Error Memo Box				
No satellites in range	No Errors				
Number Of Satellites Evaluated: 0 Number of Satellites In Range: 0		Evaluate TLE File			

**Figure 4.2. GUI Interface to the Preprocessor**

similar interfaces. The Preprocessor interface is shown in Figure 4.2. the inputs and outputs can be clearly seen and modified using this interface. It should be noted that these interfaces, although easy to use, will almost certainly not be included in the final software package to be used on board the ABL platform. They take up far too much

<b>Aircraft Parameters</b> Aircraft Latitude Degrees: 10 N Aircraft Latitude Minutes: 0 Aircraft Latitude Seconds: 0 Aircraft Longitude Degrees: 55 Aircraft Longitude Minutes: 0 Aircraft Longitude Seconds: 0 Aircraft Altitude: 12.9 Aircraft Tail ECEF X km: 0 Aircraft Tail ECEF Y km: 300 Aircraft Tail ECEF Z km: 0		<b>Orbit and Elevation Parameters</b> Look Angle: 45.0 Elev. Elevation: 45.0 Azimuth Deg: 0.2 Elevation Deg: 1.6 Azimuth Deg/Sec: 0.0001 Elevation Deg/Sec: 0.003 Max. Altitude (m): 10000 Power Pos. Limit (m): 25 Max. Pos. Limit (m): 25 Range to Max (km): 200 Other Parameters (deg): 0.0 Interpolation Interval (sec): 0.1 Max. Time to Process (sec): 2.0		<b>Simulation Time</b> Simulation Year: 1998 Simulation Month (1-12): 8 Simulation Day: 14 Simulation Hour: 3 Simulation Minute: 58 Simulation Second: 2.0 Total Duration (sec): 30	
<b>Greenwich Mean Time Parameters</b> Reference Year: 4 Reference Month: 40 Reference Second: 1.299 Modified Julian Date: 11029.0		<b>Results</b> Number of Satellites Evaluated: 0 Number of Satellites Intersected: 0 Time to Intersect: 0 Satellites Intersected (GSCN): No satellites intersected		<b>Closest Approach Results (GSCN)</b> No satellites interpolated	
<b>Error Message Box</b> No Errors		<b>Output Files</b> Input File for Preprocessor: PAData/PreprocessorOutput.txt Intersected Satellites File: PAData/IntersectingSatelliteOutput.txt Closest Approach File: PAData/ClosestApproachOutput.txt			

**Figure 4.3. GUI Interface to the Main Processor**

overhead processing to be practical, and it is likely that the PA software will be executed automatically, at laze time, not using a man/machine graphical interface. Rather, the main purpose of these interfaces is to allow an easier testing environment in which parameters can be changed quickly and output can be seen in a formatted framework. The interface to the Main processor is similar to that of the Preprocessor, and can be seen in Figure 4.3. Again, *Appendices A-F* will provide a more complete description of the modules, interfaces, inputs and outputs for the software package discussed briefly here. An analysis of the software will follow in *Chapter V*.

## **V. Analysis and Conclusions**

Now that the methodology of the Predictive Avoidance algorithm has been developed and discussed, and software has been created to automate this algorithm, we must now evaluate whether or not all of the goals of this study have been achieved. The reader will recall that the two main goals of this study were to develop a predictive avoidance algorithm, and to develop a software system that can automate this algorithm using less than 0.5 seconds during the pre-laze fire sequence. The analysis of the software in the following sections addresses how well the algorithm and software meets these goals. Areas where there may be room for improvement will also be discussed. Due to time constraints, the optimal solution to some problems encountered in this project may have been bypassed by using more convenient methods. Although it is hoped that there are not many such areas, some were a necessity to ensure this product was delivered on time. Recognizing that this thesis is only the first draft of an iterative process conducted by Boeing, it is hoped that these "lacking" areas will be further studied and refined in future iterations.

### **5.1 Software Analysis and Performance**

Numerous tests were conducted upon each software library module individually, and upon the integrated Preprocessor and Main Processor. For brevity, the individual module testing will not be discussed except to say that individual test cases were compiled to analyze each module, including extensive boundary and critical value testing. Each module was required to pass all test cases before being integrated. It must be stressed here that SGP4 was heavily relied upon, but not tested. This is the only

module that was not tested, because of its complexity, historical verification, and difficulty of independent verification and validation.

### **5.1.1 Integration Testing**

After each module was independently tested and verified, the integrated Preprocessor and Main Processor were tested using interface and boundary tests that ensured the outputs received during individual module testing were being integrated into the proper format, without corrupting data across the interfaces. Final integration testing included roughly 2,000 – 2,500 individual executions of both the Preprocessor and the Main Processor. Only approximately 100 of these runs were used to verify (via independent calculations) the correctness of the output. These 100 runs incorporated changes in the location of the platform, speed, turret rates, and times of laze. It is not beneficial to list every test here, as the software needs to be verified by an independent third party regardless. However, the performance of the software in general, and performance under some tests will be briefly discussed.

### **5.1.2 Preprocessor Software Filtering Performance**

A sample TLE satellite input file was used containing 772 unclassified satellite listings. The Preprocessor found, on average, that 22% of these input satellites were in view. By changing the location of the platform and the Universal Time of execution, testing concluded that the maximum number of satellites ever deemed in view of the platform was 209 (or 27.1%) at the location 5° Latitude, 100° Longitude, on August 14, 1998, 03:58 AM GMT. Of course, not every time and location were tested, and so this maximum percentage might rise slightly, but not much. The fewest number of satellites



seen in view is 78 (or 10.1%) at the South Pole during the same time slot. Of course, the polar caps are not of great interest, as it is unlikely that the ABL will ever see the poles, and a polar location precludes the possibility of encountering any satellites within the Geostationary Belt.

### **5.1.3 Preprocessor Software Timing Performance**

Given the input file of 772 satellites, the Preprocessor executed, on average, in 1.2 seconds Wall Clock Time (WCT). This test was conducted on a standard 200 MHz IBM compatible desktop computer, running under the Microsoft Windows 98 OS. The WCT differed between 0.9 and 1.8 seconds, depending upon the CPU load exerted by other applications at the time of the test. WCT will depend heavily upon the system used to run the Preprocessor. Although the Preprocessor is not required to run within a given time budget, it is good to know that it is fairly quick and can be run multiple times in a minute, if needed. The graphical interface and the input/output files constitute a large part of this WCT. It is clear that WCT will be substantially reduced when the GUI is stripped off, and the I/O is handled in memory rather than through disk read/writes.

### **5.1.4 Main Processor Software Filtering Performance**

For a given sample test, where the platform was located at 0° Latitude, 0° Longitude, the sample TLE satellite input file was used, containing 772 unclassified satellite listings. This is the file that the Main Processor would encounter if no Preprocessing was accomplished ahead of time. The Main Preprocessor found that 17 (or 2.2%) of these input satellites were close enough to interpolate (17 satellites were interpolated). By changing the location of the platform and the Universal Time of

execution, testing concluded that the maximum number of satellites ever close enough to be interpolated was 18 (Or 2.3% of the active satellite file), so this sample test is approaching the maximum number of close-approach satellites that have been seen in any test. Again, not every time and location were tested, and so this maximum number may be exceeded somewhere, but not by much. When preprocessing is performed, the input file to the Main Processor drops to 198 satellites (the satellites that are in view), which is still a somewhat high concentration. When the Main Processor analyzed this new input file, it found 16 (rather than the expected 17) close-approach satellites. It is expected that the number of close-approach satellites should not change, because preprocessing only strips away satellites that could not possibly intersect the laser. The reason for this discrepancy is that one of the 17 satellites that was forecast to intersect the laser was on the other side of the Earth, but during the exact moment of the forecast was accelerating at a very fast rate toward the laser. When this fast acceleration (which in reality lasts only fractions of a second) is propagated 30 seconds into the future, it results in an unrealistic forecast that is quickly weeded out using interpolation. At no time during random testing was an actual intersection recorded. Rather the intersection tests had to be engineered and manipulated by the tester, because of the extremely low probabilities of an actual intersection. There were no test cases developed, engineered or random, that could produce more than one intersected satellite.

#### **5.1.5 Main Processor Software Timing Performance**

Given the unprocessed input file of 772 satellites, the Main Processor executed, on average, in 0.9 seconds WCT. This test was conducted on the same computer as the Preprocessor. When given the preprocessed file of 198 in-view satellites, the Main

Processor execution time dropped down to 0.25 seconds WCT. Again, WCT will be substantially reduced when the GUI is stripped off, and the faster memory I/O is used. The reader will recall that the requirement for execution is 0.5 seconds, so there is plenty of room for additions/modifications to the software.

## **5.2 Further Study**

There are a number of areas that may require further study, and have not been thoroughly considered in this thesis. It is hoped that these topics will be addressed during future iterations in the development of the final software package to be used with the ABL platform. These topics are only briefly addressed here.

### **5.2.1 Missile Tracking**

Currently, the software treats the missile trajectory as a static entity, with initial parameters that do not change. This is reflected in the laser turret position, velocity and acceleration variables which are read at forecast time, and held constant throughout the forecast duration, as long as thirty seconds. It is unreasonable to believe that these parameters cannot vary while the missile is in the boost phase. Different ballistic missile systems have burn rates that vary significantly throughout the duration of the burn. While the initial acceleration may not change, there is also a good probability that it will change. A significant change from initial conditions will, of course, invalidate a forecast that is based on those initial conditions. There are at least two feasible methods to counter this variance from initial conditions. The first method is to store another data file that accurately describes the burn rates of all possible missile targets, and use this information to more accurately predict missile trajectory. The second suggested method

is to simply rerun the Main Processor when and if the actual conditions change from the initial conditions by some slight error angle, and then include that slight error angle in the forecast error angle already computed for the scenario. Using this method, the Main Processor would re-execute during the laze *only if* the initial conditions are seriously compromised by the actual missile trajectory. Although there is an *extremely remote* possibility that this recomputation could result in laze interruption, this is unlikely, given the fact that 2000 random runs of the Main Processor have not produced a single intersection yet. Even if an intersection was predicted, the decision to terminate the laze still rests in comparing the importance of the target to the importance of the compromised satellite. This second solution's strength is that it is easily implemented and tested.

### **5.2.2 Atmospheric Refraction**

As laser energy travels through the atmosphere, it encounters differing atmospheric densities until it reaches the relatively empty vacuum of space. As light encounters these differing densities, the index of refraction of the medium (air) changes, causing the leading edge of the beam to travel at a slightly different speed than the trailing edge. Over a large distance, this can cause the beam to arc (or refract) slightly until it reaches space. Quick calculations show that this arc may cause as much as a 0.4 degree change within the atmosphere for small slant angles traveling a long distance before hitting the vacuum of space. This degree change would then be further added to as the beam propagates at its new trajectory through space. This refractive issue is a challenge that was encountered too late to incorporate into this iteration, but it is still extremely important. The solution is, of course, to account for this refraction given the location of the platform and the starting parameters of the laser turret. Although the

refractive index of air varies from location to location, for our purposes, it can be held constant over given altitudes with only minor errors in the refraction calculation.

### **5.2.3 Error Angle Determination**

Currently, the error angle for a given satellite encounter is computed using rough position error approximations. This results in an error angle that is an absolute. That is, we can either hit it or miss it. Further the error angle describes a spherical comfort zone around the satellite when in fact, the error ellipsoid should probably be more oblong and eccentric given the types of errors encountered. One possible solution to further define the error ellipsoid is to use covariance matrices to model each of the position error ellipsoids, resulting in a more probabilistic definition of the comfort zone around the satellite that we wish to avoid. The difficulties with this solution lie in the population of each covariance matrix, and the definition of exactly what probability is "too high" a risk when describing the approach of the laser to the satellite. Because this solution would have added a significant time cost with marginal fidelity improvement, it was decided to forego this method in favor of the simpler half-error angle approach.

### **5.2.4 Forecast/Interpolation Fine Tuning**

Currently the Main Processor does a rough forecast that interprets more satellites being intersected that are actually intersected, and then interpolates the position of each "intersecting" satellite to ensure that it actually does intersect. During testing of the Main Processor, no satellites that approached close enough to the laser to possibly be threatened were ever "not caught" by the initial forecast. This is not to say that it could not happen however. Independent calculations have shown that, theoretically, a small

possibility exists that a LEO range satellite with a large error angle could possibly slip by if the satellites comfort zone (error angle) is just barely touched by the laser turret angle in the first 0.3 seconds after the forecast is conceived. In this case, there is a small possibility that the initial conditions would be derived near the vertex, and that the forecast closest approach angle is predicted to occur long after it actually has occurred. This is a problem because there is a set "Interpolation Buffer Time" before the forecast closest approach starts when the interpolation begins. If the time between the forecast closest approach and the actual closest approach is greater than this buffer time, then interpolation will not catch an intersection that occurs at the actual closest approach time. This has not occurred in any test cases, and attempts to engineer such a case failed repeatedly due to the time and precision needed to model such an approach. This is not to say, however, that it is impossible. A possible solution might be to increase the buffer, which only slightly affects the overall run-time (depending upon the step size). Further, attempts should be made to refine the interpolation step size. During testing, it became evident that 0.1 seconds was adequate for the precision needed, without compromising too much efficiency. This is not to say, however, that 0.1 seconds is the optimum size, nor that it will catch every possible intersection.

### **5.2.5 Software Speed and Testing**

As mentioned previously, this software was not necessarily written to have the fastest possible execution time. Rather, the primary goal was to write the software so that it is easily understandable. This often involved coding an algorithm using an inefficient implementation so that it matched (both functionally and visually) the methodology as it is presented in this thesis, rather than using an elegant but fuzzy solution. Because this

code was designed and implemented with the understanding that it would be just the first step of an iterative solution, understandability was considered the highest priority to ensure a successful handoff. Testing is likewise considered to be iterative. Although considerable in-house testing was completed by the author, it was a one-man effort. One man does not make a very dynamic testing team, especially when that one man is the designer, coder, and verification/validation checker! That man, though disciplined, is bound to have his own biases and iterative mistakes that cannot be cross-checked with anyone else. Therefore the value of the testing conducted by him is diminished, and mistakes are likely to still exist.

### **5.3 Conclusion**

There has been much ground covered here, and it is hoped that the Predictive Avoidance algorithm and its corresponding software will prove to be useful in future modeling efforts. Although this thesis applies to a narrow application platform, namely the ABL, it can be seen that the general algorithm is broadly applicable to a wide range of directed-energy targeting applications. For instance, setting the platform speed and altitude to zero will result in a land-based model for predictive avoidance. It is likely that directed-energy weapons will become more widespread if the technology can be adequately exploited with the first operational ABL series. If so, the general principles tied together in this thesis will find broader application.

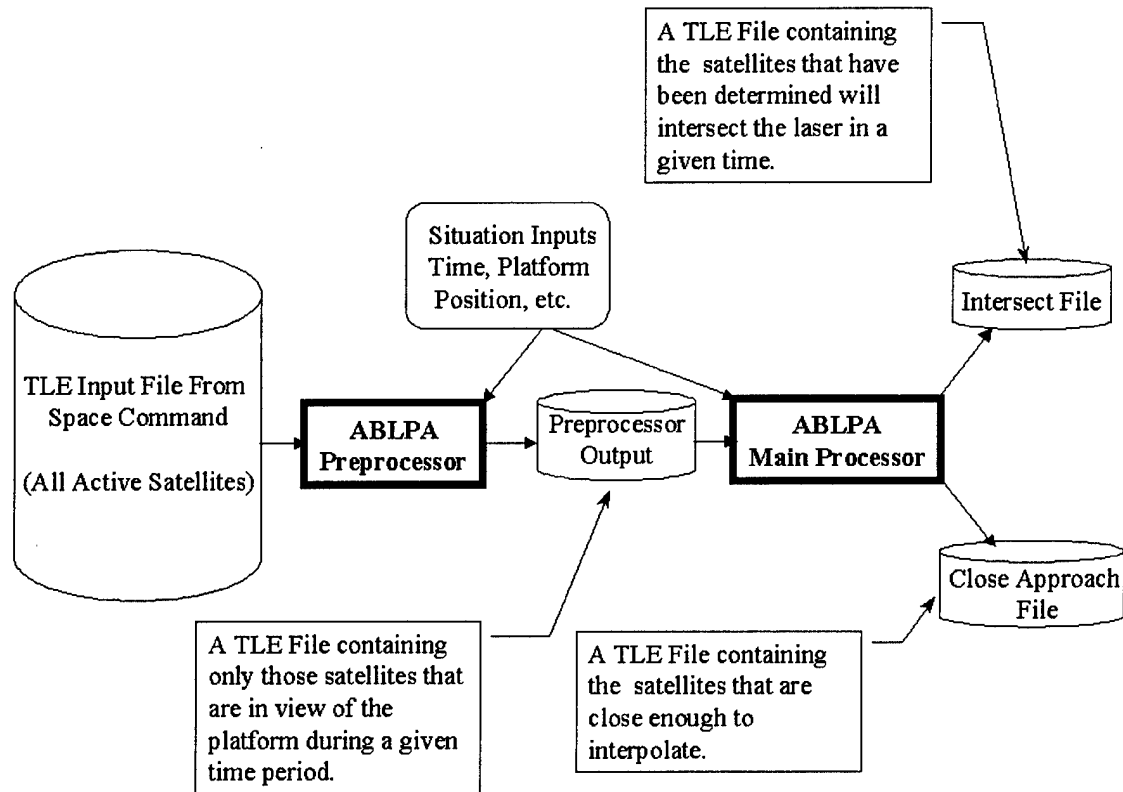
## **Appendix A – The Software Structure**

The software implementation by which the algorithms discussed previously can be automated is discussed in this appendix. This appendix, combined with the following appendices, comprise a rough software programmer's manual. The software has been written in a modular format, and has been broken into logically functional modules that have been designed to work both together and independently, if needed. As mentioned in *Chapter IV*, this software package has been designed with three conflicting objectives. The first objective is to make the software readily understandable to a person who wishes to study it in the future. The second goal is that the software be fast. It is estimated that the prediction avoidance software should not need more than 0.5 seconds to fully process a mission. The third major goal for the PA software is that it should be modular. Of these three goals for the software, understandability is the by far the most important. There are many cases within the software in which a fluid, slow, understandable implementation has been used instead of a speedier vague implementation. This is done with the understanding that the software will be reviewed at a later time, when any "slow" algorithms may be supplanted with the software engineer's choice of implementations. The standard flow of the software is shown in Figure A.1.

### **A.1 Modularity and Testing**

As mentioned previously, modularity is a goal for this software package. Although the software package has been broken into two separate entities, namely the Preprocessor and the Main Processor, the modules that compose this project have been grouped together into twelve smaller libraries, by their logical functionality. The reason





**Figure A.1. The Predictive Avoidance Software Flow**

for this breakdown is simple. The project as a whole is difficult to test as a whole. Breaking down the project into twelve testable mini-projects allows independent testing of a portion of the software while divorcing it from the whole. The benefits to this are intuitively obvious. Understandability and testability are increased, however, creation of this type of infrastructure in the project has also required that the code be that much larger, with more physical modules and interfaces. Each library is distinguished by having its own stand-alone GUI that calls the module(s) being used in that library. As implied before, there will be twelve software libraries total. Two of these will be the final ABLPA Preprocessor and Main Processor. The other ten will be composed of lower and lower levels of subordinate modules; five testing modules in the Preprocessor, and five more introduced in the Main Processor.

## A.2 The Calculation Modules

The modules that house the meat of the algorithm and the calculation intensive software are written in the C++ language. The compiler used is Borland C++ Builder 3 (Standard, C++ Version 5). It is likely that the code for these modules can be recompiled with minimal effort using another similar C++ compiler. These modules are written for easy adaptability to other environments. All of the code for the calculation modules will be included and discussed, as its explanation and operation is one of the important products that this thesis delivers. *Appendix B - The ABLPA Preprocessor Software* will discuss all modules developed for the ABLPA Preprocessor, along with their corresponding interface parameters. *Appendix C - The ABLPA Main Processor Software* will discuss modules developed for the Main Processor that have not already been covered in *Appendix B*. The actual implementation code listings for the calculation modules is given in *Appendix D – Implementation Code*.

## A.3 The Test Modules for Each Software Library

There are other modules that have been developed exclusively for the task of calling and interfacing with the Preprocessor the Main Processor, and each of the other ten libraries. They are strictly “front-end” interfaces for the calculation modules that can be used to run and test the algorithms that have been developed within the calculation modules. These test modules have been kept as “simple” as possible, while still maintaining ease-of-use, to avoid the necessity of “testing” the test modules. They consist of a graphical interface that collects input to the module(s) being tested, and the function call to those modules. The graphical nature of the test modules utilizes quite a

bit of compiler-specific organization and terminology to allow easy development of these modules. Therefore, any change or recompilation of these graphical interface modules will require the user to have a copy of Borland C++ Builder 3 (Standard). This compiler will also be required if the user of this software wishes to run the fully compiled software packages that have included within them the GUI interfaces. The Test Modules and their software code is described further in *Appendix E – Test Module Code*.

#### **A.4 The Environment**

These modules have all been developed using a standard desktop IBM compatible computer, using a 200 MHz Intel Pentium processor. The modules are compiled to run in the Microsoft Windows 95 or 98 operating system environment. Operation in other environments has not been tested and is not guaranteed, especially the graphical test modules.

#### **A.5 Test Module Example**

As mentioned previously, the product, C++ Builder 3, made by Borland was used to craft a graphical front-end to each library designed. One GUI has been created for each functional task (or library) needed in the Preprocessor and Main Processor. Each of these modules, where practical, comes with a graphical front-end interface to allow testing of individual components. This test module example is included to give the user an idea of the design of each front-end. Each front-end is designed simply, without contributing to the solution of the algorithm handled by the module being called. Because each of these test modules have little worth in themselves, I will demonstrate

how just one of the test modules work, and only include a cursory graphical figure when describing test modules in the future. The module chosen to describe the interface here is the module that interfaces directly with the SGP4 time propagation modules. This GUI routine tests modules in the library “SGP4Support”. The main calculation module (“CallSGP4”) is held within “SGP4SupportModules.cpp”. This module, in turn, calls the SGP4 routines created by Air Force Space Command, which are held in “SGP4Routines.cpp”. The test module Graphical User Interface (GUI) used to call this main routine is called “SGP4TestForm”, held in a physical module of the same name. The graphical interface is shown in Figure A.2.

The screenshot displays the 'SGP4 Support' GUI interface, which is divided into several sections:

- SATELLITE INFORMATION:** This section contains two columns of input fields. The left column includes SSC Number (0), Classification (0), International ID (0), Epoch Year (1998), Epoch Day (216.91470564), Revs /Day Squared (0), Revs /Day Cubed (0), BStar Drag (0), and Ephemeris Type (0). The right column includes Element Set Number (0), Inclination (45.4208), Right Ascension (0.1433), Eccentricity (0.0005988), Argument of Perigee (77.1974), Mean Anomaly (262.9894), Mean Motion (14.5700563), Rev At Epoch (0), and Julian Date to Prop (11029.1906229).
- Populate Using File:** A button labeled 'Populate Using File' is next to a text field containing 'PAData/LEOFile1.txt'.
- Run SGP4 (Version 3.01):** A large button spanning the width of the input fields.
- TESTING CRITERIA:** A column of ten input fields on the right side, each with a label and a value of 0: X (km), Y (km), Z (km), Xdot (km/sec), Ydot (km/sec), Zdot (km/sec), Delta Time (Min), Inclination, Right Ascension, Eccentricity, Mean Motion, Arg of Perigee, and Mean Anomaly.
- Error Messages:** A section at the bottom left with the text 'No Errors' and a large empty text area below it.

**Figure A.2. GUI Interface to the “SGP4 Support” Project**

### **A.5.1 Description of Code**

The code for SGP4TestForm is only partially included here. A software programmer will notice that much of the code infrastructure is missing. This is because C Builder handles much of the behind-the-scenes programming and leaves only the event-handlers for implementation by the programmer. So, in reality only the event-handlers are shown in the code that follows. An “event-handler” is anything that can be manipulated. For instance, any button that can be “pushed” on the GUI may have a small routine, called an event-handler, which executes a set of instructions. Having only the event handlers in a condensed piece of code allows the maintainer to easily grasp and change the nature of the GUI. There are a few things to point out in the code that follows. First, there are two main event-handlers associated with this GUI. Their names are “FileButtonClick” and “RunButtonClick”. This is appropriate because the GUI template in Figure A.1 has exactly two buttons that can be pushed. The First button is the “Propagate Using File” button that activates the “FileButtonClick” event-handler, the second is the “Run SGP4 (Version3.01)” button that activates the “RunButtonClick” event-handler. Notice that each routine is fairly simple and straight-forward. There is no attempt to solve any part of the Predictive Avoidance algorithm within the event-handler itself. Rather, they simply make a call to the routines that do the work. The sole purpose of the event-handler is to take inputs, call a calculation module, and format the resulting output. For instance, RunButtonClick simply calls the module “CallSGP4”. FileButtonClick calls “ReadTLEFile”. So these test GUIs can be seen to be simply interfaces that allow easy, extensive testing of the calculation modules, which house the meat of the project. The code follows on the next few pages.

## A.5.2 Code Listing for SGP4TestForm

```

/*****
/*  MODULE NAME:      SGP4TestForm.cpp
/*  AUTHOR:          Captain David Vloedman
/*  DATE CREATED:    October 10, 1998
/*
/*  PURPOSE:         This test form module is a test module for the routines */
/*                   handle calling of the satellite propagator. "SGP4". This*/
/*                   propagator is US Space Command's satellite time/position*/
/*                   propagator using general perturbations only. The
/*                   version of SGP4 used here is version 3.01 in C
/*
/*
/*  COMPILER:        Borland C++ Builder3 Standard version
/*                   This compiler should be used to compile and link.
/*
*****/
/*****
/* C++BUILDER-SPECIFIC LIBRARIES */
/*****
#include <vcl.h>
#pragma hdrstop
#pragma package(smart_init)
#pragma resource "*.dfm"
/*****
/* USER-BUILT LIBRARIES
/*
/*****
#include "SGP4TestForm.h"
#include "SGP4SupportModules.h"
#include "LaserConstants.h"
#include "Satellite.h"
#include "ErrorStructure.h"
#include "TLEInput.h"
/*****
/* C STANDARD LIBRARIES
/*
/*****
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <iostream.h>
#include <conio.h>
#include "SGP4Routines.h"
#include "TimeModules.h"
/*****
/*      CREATE THE FORM
/*
/*****
TForm1 *Form1;

//-----
__fastcall TForm1::TForm1(TComponent* Owner)
: TForm(Owner)
{
}
/*****
/*  THIS EVENT HANDLER PROCEDURE HANDLES THE BUTTON*/
/*  THAT CAN LOAD A TEST CASE FROM A FILE FOR LATER*/
/*  EXECUTION
/*
/*****
void __fastcall TForm1::FileButtonClick(TObject *Sender)
{

```

```

ErrorStructure ErrorList;
SatStructure *SatArray = new SatStructure;

char Errors[MAXERRORS][MAXMESSAGELENGTH];
int i;
ErrorStructure *ErrorPtr=&ErrorList; /* A POINTER TO ERRORLIST */
char FileName[MAXNAMELENGTH] = " ";

/*****
/* GET NAME OF FILE TO READ TEST CASE FROM */
*****/
strcpy(FileName,FileEdit->Text.c_str());

/*****
/* READ ALL SATELLITES FROM THE FILE, AND USE THE */
/* FIRST SATELLITE IN THE FILE AS THE TEST CASE */
*****/
ReadTLEFile(FileName,
            *SatArray,
            *ErrorPtr);
/*****
/* NOTE THE Sat[0] IS THE FIRST SATELLITE IN THE */
/* FILE */
*****/
SSCEdit->Text = String(SatArray->Sat[0].GetSSCNumber());
ClassEdit->Text = String(SatArray->Sat[0].GetSecurityClass());
IntIDEdit->Text = String(SatArray->Sat[0].GetInternationalID());
EpochYearEdit->Text = String(SatArray->Sat[0].GetEpochYear());
EpochDayEdit->Text = String(double(SatArray->Sat[0].GetEpochDay()));
RevSquaredEdit->Text = String(double(SatArray->Sat[0].GetRevSquared()));
RevCubedEdit->Text = String(double(SatArray->Sat[0].GetRevCubed()));
BStarEdit->Text = String(double(SatArray->Sat[0].GetBStarDrag()));
EphemerisTypeEdit->Text = String(SatArray->Sat[0].GetEphemerisType());
ElSetEdit->Text = String(SatArray->Sat[0].GetElementSetNumber());
InclinationEdit->Text = String(double(SatArray->Sat[0].GetInclination()));
RightAscensionEdit->Text=String(double(SatArray-
>Sat[0].GetRightAscension()));
EccentricityEdit->Text = String(double(SatArray-
>Sat[0].GetEccentricity()));
ArgumentOfPerigeeEdit->Text = String(double(SatArray-
>Sat[0].GetArgumentOfPerigee()));
MeanAnomalyEdit->Text = String(double(SatArray->Sat[0].GetMeanAnomaly()));
MeanMotionEdit->Text = String(double(SatArray->Sat[0].GetMeanMotion()));
RevNumberEdit->Text = String(SatArray->Sat[0].GetRevAtEpoch());

/*****
/* DISPLAY ALL ERRORS */
*****/
CreateDisplayText(ErrorList, Errors);
if (ErrorList.TotalErrors()!=0)
{
    ErrorMemoBox->Lines->Clear();
    ErrorMemoBox->Lines->Add("THERE ARE ERRORS...");
    for (i = 0; i<ErrorList.TotalErrors(); i++)
        ErrorMemoBox->Lines->Add(Errors[i]);
}
else
{
    ErrorMemoBox->Lines->Clear();
    ErrorMemoBox->Lines->Add("No Errors...");
}

```

```

}

/*****
/* THIS PROCEDURE ACTUALLY RUNS THE TEST CASE AS */
/* IT HAS BEEN ENTERED INTO THE FORM AND DISPLAYS */
/* THE RESULTS OF THE RUN */
*****/
void __fastcall TForm1::RunButtonClick(TObject *Sender)
{
    ErrorStructure ErrorList;
    ErrorStructure *ErrorPtr=&ErrorList; /* A POINTER TO ERRORLIST */
    Satellite* Sat;
    Sat = new Satellite;
    char Errors[MAXERRORS][MAXMESSAGELENGTH];
    int i;
    char buff[MAXNAMELENGTH];
    double JulianDate;
    double Inclination;
    double *InclinationPtr = &Inclination;
    double RightAscension;
    double *RightAscensionPtr = &RightAscension;
    double Eccentricity;
    double *EccentricityPtr = &Eccentricity;
    double MeanMotion;
    double *MeanMotionPtr = &MeanMotion;
    double ArgumentOfPerigee;
    double *ArgumentOfPerigeePtr = &ArgumentOfPerigee;
    double MeanAnomaly;
    double *MeanAnomalyPtr = &MeanAnomaly;
    double X;
    double *XPtr = &X;
    double Y;
    double *YPtr = &Y;
    double Z;
    double *ZPtr = &Z;
    double Xdot;
    double *XdotPtr = &Xdot;
    double Ydot;
    double *YdotPtr = &Ydot;
    double Zdot;
    double *ZdotPtr = &Zdot;
    double Delta;
    double *DeltaPtr = &Delta;

    /*****
    /* GET SATELLITE EPHEMERIS INFORMATION */
    *****/
    Sat->SetSSCNumber(SSCEdit->Text.ToInt());
    strcpy(buff,ClassEdit->Text.c_str());
    Sat->SetSecurityClass(buff);
    strcpy(buff,IntIDEdit->Text.c_str());
    Sat->SetInternationalID(buff);
    Sat->SetEpochYear(EpochYearEdit->Text.ToInt());
    Sat->SetEpochDay(EpochDayEdit->Text.ToDouble());
    Sat->SetRevSquared(RevSquaredEdit->Text.ToDouble());
    Sat->SetRevCubed(RevCubedEdit->Text.ToDouble());
    Sat->SetBStarDrag(BStarEdit->Text.ToDouble());
    Sat->SetEphemerisType(EphemerisTypeEdit->Text.ToInt());
    Sat->SetElementSetNumber(ElSetEdit->Text.ToInt());

```



```

Sat->SetInclination(InclinationEdit->Text.ToDouble());
Sat->SetRightAscension(RightAscensionEdit->Text.ToDouble());
Sat->SetEccentricity(EccentricityEdit->Text.ToDouble());
Sat->SetArgumentOfPerigee(ArgumentOfPerigeeEdit->Text.ToDouble());
Sat->SetMeanAnomaly(MeanAnomalyEdit->Text.ToDouble());
Sat->SetMeanMotion(MeanMotionEdit->Text.ToDouble());
Sat->SetRevAtEpoch(RevNumberEdit->Text.ToInt());
JulianDate = JulianDateEdit->Text.ToDouble();

/*****
/* MAKE A CALL TO THE SGP4 PROPAGATOR */
*****/
CallSGP4(*Sat,
        JulianDate,
        *XPtr,
        *YPtr,
        *ZPtr,
        *XdotPtr,
        *YdotPtr,
        *ZdotPtr,
        *InclinationPtr,
        *RightAscensionPtr,
        *EccentricityPtr,
        *MeanMotionPtr,
        *ArgumentOfPerigeePtr,
        *MeanAnomalyPtr,
        *DeltaPtr,
        *ErrorPtr);

/*****
/* Convert Mean Motion from radians/sec to */
/* revolutions per day */
*****/
MeanMotion = MeanMotion * MINUTESPERDAY / TWOPI;

/*****
/* DISPLAY THE RESULTS OBTAINED FROM SGP4 */
*****/
XEdit->Text = String(X);
YEdit->Text = String(Y);
ZEdit->Text = String(Z);
XdotEdit->Text = String(Xdot);
YdotEdit->Text = String(Ydot);
ZdotEdit->Text = String(Zdot);
DeltaEdit->Text = String(Delta);
InclinOutEdit->Text = String(Inclination);
RightAsOutEdit->Text = String(RightAscension);
EccentricityOutEdit->Text = String(Eccentricity);
MeanMotionOutEdit->Text = String(MeanMotion);
ArgOfPerigeeOutEdit->Text = String(ArgumentOfPerigee);
MeanAnomalyOutEdit->Text = String(MeanAnomaly);
DeltaEdit->Text = String(Delta);

/*****
/* DISPLAY ALL ERRORS */
*****/
CreateDisplayText(ErrorList, Errors);
if (ErrorList.TotalErrors() != 0)
{
    ErrorMemoBox->Lines->Clear();
    ErrorMemoBox->Lines->Add("THERE ARE ERRORS...");
}

```

```

        for (i = 0; i<ErrorList.TotalErrors(); i++)
            ErrorMemoBox->Lines->Add(Errors[i]);
    }
    else
    {
        ErrorMemoBox->Lines->Clear();
        ErrorMemoBox->Lines->Add("No Errors...");
    }
}
//-----

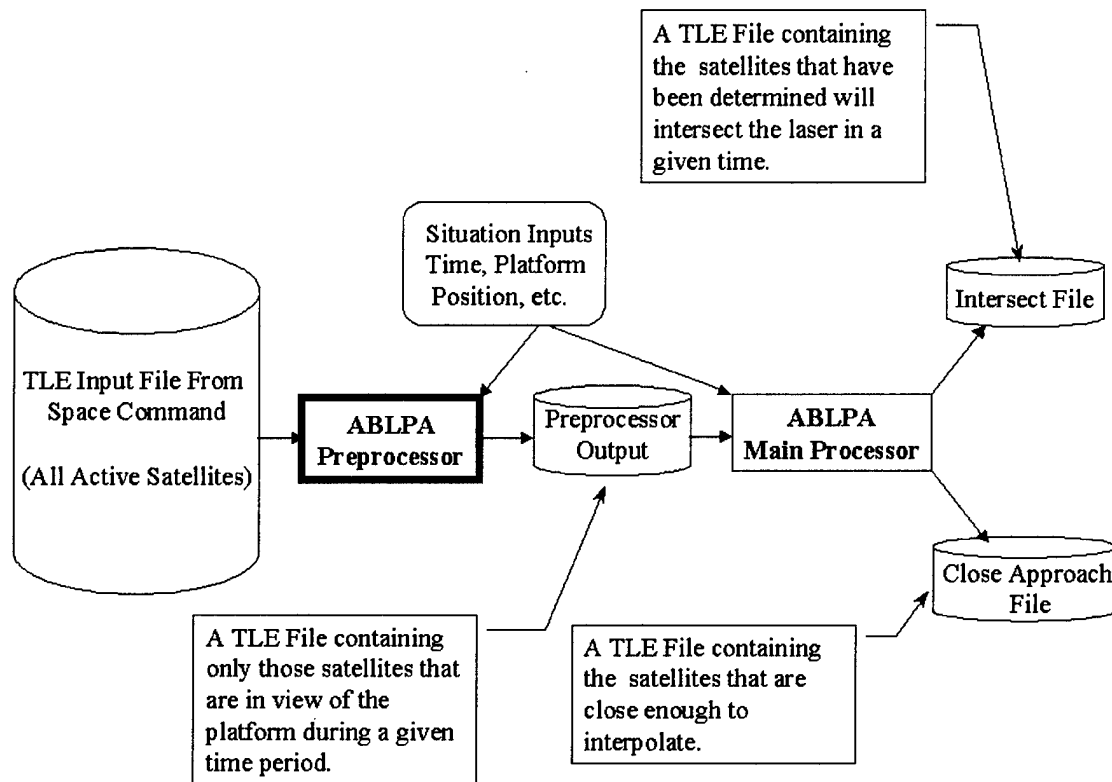
```

## A.6 Error Handling

Each of the event-handlers in the code listed above can be seen to have an error handling routine that lists out all errors that have been trapped by the program. It is important that the structure of this error-handling be known for any programmers in the future who may wish to adopt all or part of the coded modules of this project. Each module within both the ABLPA Preprocessor and the ABLPA Main Processor have an extra parameter in their parameter list that holds and traps any errors handled by the program. This error parameter is always the last parameter on the visible parameter list and can only be accessed by the manipulation routines held in the module "ErrorStructure". These routines and the nature of the error-handling system are discussed in greater detail further in this paper.

## Appendix B. ABLPA Preprocessor Software Implementation

The Airborne Laser Predictive Avoidance (ABL-PA) Preprocessor is only a part of the software developed in this project. Figure B.1 illustrates how this preprocessor fits into the overall hierarchy of the software.

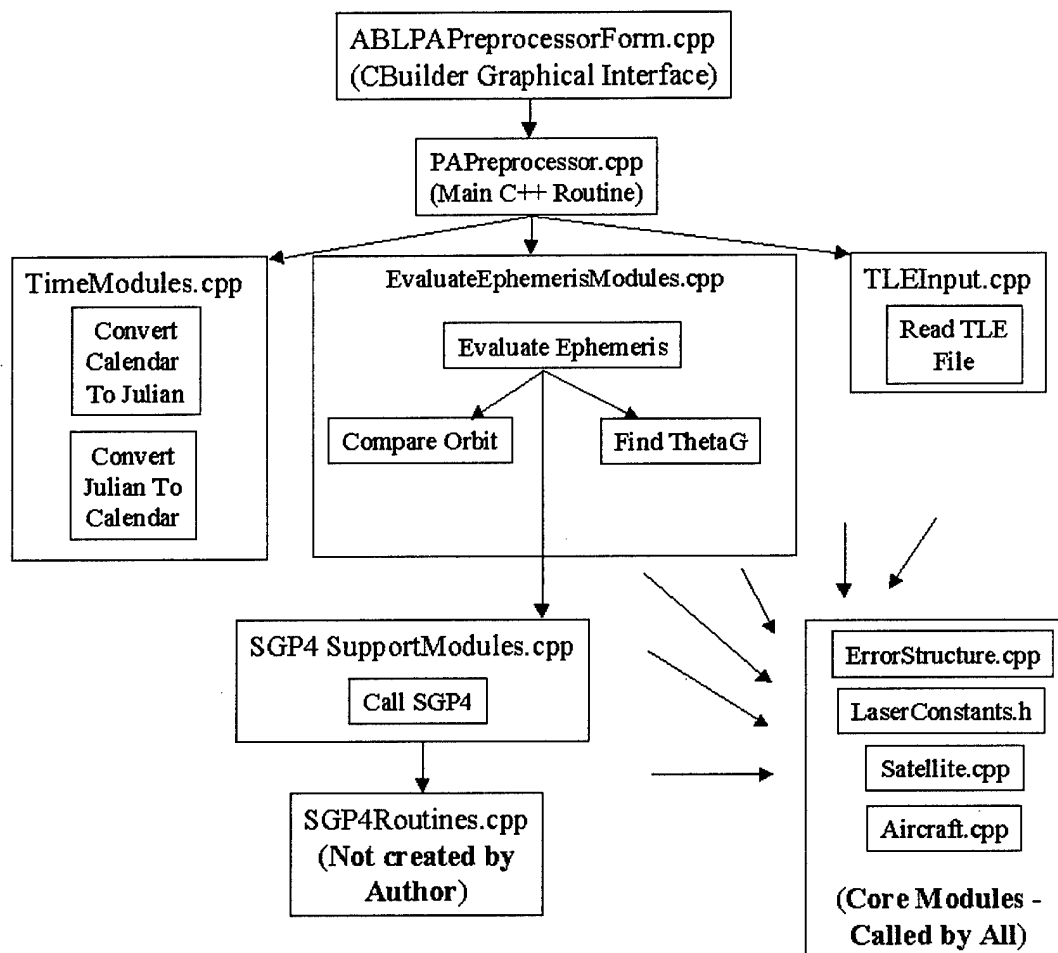


**Figure B.1 Where the ABLPA Preprocessor Fits in the Software Hierarchy**

It can be seen that the task of the preprocessor is to take the input file containing all active satellites in orbit, and strip out only those satellites that are in view of the laser platform at a given time. The preprocessor then creates an output file containing the satellites in view. This output file has exactly the same format as the main TLE file, except it has fewer satellites within it.

## B.1 Preprocessor Modular Format

The preprocessor is a conglomeration of many software libraries that were created and tested independently before being combined to form the preprocessor. Figure B.2 shows the basic libraries that comprise the preprocessor, and the modules each library contains. Each library and module shown will be explained within this chapter.



**Figure B.2 ABLPA Preprocessor Calling Tree**

From Figure B.2 it can be seen that modules could be roughly grouped into six libraries. Each of these is listed in Table B.1. Each of these libraries of modules has been designed to be a project in and of themselves, tested using their own GUI as seen in the table.

**Table B.1 The Six Libraries Composing the ABLPA Preprocessor**

<b>Software Library Title</b>	<b>Modules Tested (C++)</b>	<b>GUI Interface Module (C++ Builder 3)</b>	<b>Purpose</b>
<b>ABLPA Preprocessor</b>	All preprocessor modules as shown in <i>Figure B.2</i>	PAPreprocessorForm	To provide a user-friendly way to run the preprocessor
<b>Test Error Structure</b>	ErrorStructure.cpp CreateDisplayText AddError GrabError	TestErrorStructure Text Only – Non-(Graphical)	To test the error handling routines used to record and store errors
<b>SGP4 Support</b>	CallSGP4 SGP4Routines Core Modules	SGP4TestForm	To test the interface with SGP4, as well as the output received from the propagator
<b>Time Module Test</b>	ConvertJulianToCalendar ConvertCalendarToJulian Core Modules	TimeTestForm	To test the time conversion modules
<b>TLE Input Test</b>	ReadTLEFile Core Modules	TLETestForm	To test the module that reads the Two-Line Element Set files used by the software
<b>Test Evaluate Ephemeris</b>	EvaluateEphemeris CompareOrbit FindThetaG Core Modules	EvaluateEphemerisForm	To test the evaluation of a single satellite

The discussion of the preprocessor will progress through each of these libraries individually, discussing the nature of the function served by the library, as well as comments on each module within that library. The interfaces and input/output parameters used with each module will be emphasized. The actual code for each module in the ABLPA Preprocessor will be listed out in *Appendix D*. The code for each sub-

project GUI interface will be listed in *Appendix E*. Only the "Header File" or the files with the ".h" extension will be listed here in the discussion, because they are short and contain important interface information that should be discussed. All of the implementation code will be included in their respective Appendices.

## B.2 The Core Modules

The Core Modules are modules that are used extensively throughout both the Preprocessor and the Main Processor. They consist of the modules, Aircraft.cpp, Satellite.cpp, LaserConstants.h, and ErrorStructure.cpp. Except for ErrorStructure.cpp, none of the Core Modules are tested exclusively, because their design is fairly simple, and their function is easily recognized.

### B.2.1 Aircraft.h

This module defines the "Aircraft" object. This object, or data-type, is used to store all information needed about the aircraft platform parameters, including position, speed, and etc. Its header file, which follows, describes the various manipulation functions defined to work with the Aircraft object. If more clarification is required, the implementation listing in *Appendix D* may help to clarify.

```

/*****
/*  MODULE NAME:      Aircraft.h
/*  AUTHOR:          Captain David Vloedman
/*  DATE CREATED:     Sept 19, 1998
/*
/*  PURPOSE:         This module of code houses the Aircraft class object.
/*
/*  COMPILER:        Borland C++ Builder3 Standard version
/*                   This compiler should be used to compile and link.
/*
*****/

#ifndef AircraftH
#define AircraftH

#include "LaserConstants.h"

```

```

class Aircraft {
    public:
        Aircraft();
        ~Aircraft();

    /**
     * AIRCRAFT MANIPULATION FUNCTIONS
     */
    void SetLatitudeDegree(int ld);
    void SetLatitudeMinute(int ld);
    void SetLatitudeSecond(double ls);
    void SetLatitudeHemisphere(int h);
    void SetLongitudeDegree(int ld);
    void SetLongitudeMinute(int ld);
    void SetLongitudeSecond(double ls);
    void SetVelocityX(double vel);
    void SetVelocityY(double vel);
    void SetVelocityZ(double vel);
    void SetAltitude(double alt);

    int    GetLatitudeDegree();
    int    GetLatitudeMinute();
    double GetLatitudeSecond();
    int    GetLatitudeHemisphere();
    int    GetLongitudeDegree();
    int    GetLongitudeMinute();
    double GetLongitudeSecond();
    double GetVelocityX();
    double GetVelocityY();
    double GetVelocityZ();
    double GetAltitude();

    private :
        int    LatitudeDegree;
        int    LatitudeMinute;
        double LatitudeSecond;
        int    LatitudeHemisphere;
        int    LongitudeDegree;
        int    LongitudeMinute;
        double LongitudeSecond;
        int    VelocityX;
        int    VelocityY;
        int    VelocityZ;
        double Altitude;
};

#endif

```

### B.2.2 Satellite.h

This module defines the "Satellite" object. This object, or data-type, is used to store all information needed about the satellite ephemeris parameters. Its header file, which follows, describes the various manipulation functions defined to work with the Satellite object. If more clarification is required, the implementation listing in *Appendix D* may help to clarify.

```

/*****
/*  MODULE NAME:      Satellite.h
/*  AUTHOR:          Captain David Vloedman
/*  DATE CREATED:     July 25, 1998
/*
/*  PURPOSE:         This module of code houses the Satellite class object.
/*
/*  COMPILER:        Borland C++ Builder3 Standard version
/*                   This compiler should be used to compile and link.
/*
*****/

#ifndef SatelliteH
#define SatelliteH

#include "LaserConstants.h"

class Satellite {
public:
    Satellite();
    ~Satellite();

/*****
/*  SATELLITE MANIPULATION FUNCTIONS.  MANY OF THESE
/*  FUNCTIONS ARE BASED ON THE FIELDS OF THE TWO-LINE
/*  ELEMENT SET INPUT FORMAT USED BY SPACE COMMAND.
*****/
    void SetName(char name[MAXINPUTLINELENGTH]);
    void SetSSCNumber(long int ssc);
    void SetRevAtEpoch(long int rev);
    void SetSecurityClass(char secclass[CLASSLENGTH+1]);
    void SetInternationalID(char intID[INTNUMLENGTH+1]);
    void SetEpochYear(int eyear);
    void SetEphemerisType(int etype);
    void SetElementSetNumber(int esetnum);
    void SetEpochDay(long double eday);
    void SetRevSquared(long double rev2);
    void SetRevCubed(long double rev3);
    void SetBStarDrag(long double bstar);
    void SetSemiMajorAxis(long double sma);
    void SetEccentricity(long double e);
    void SetRightAscension(long double ra);
    void SetInclination(long double i);
    void SetArgumentOfPerigee(long double ap);
    void SetMeanAnomaly(long double ma);

```



```

void SetEccentricAnomaly(long double ea);
void SetTrueAnomaly(long double ta);
void SetScalarRadius(long double sr);
void SetMeanMotion(long double mm);
void Satellite::SetTLELine1(char line[MAXINPUTLINELENGTH]);
void Satellite::SetTLELine2(char line[MAXINPUTLINELENGTH]);

```

```

char*      GetName();
long int   GetSSCNumber();
long int   GetRevAtEpoch();
char*      GetSecurityClass();
char*      GetInternationalID();
int        GetEpochYear();
int        GetEphemerisType();
int        GetElementSetNumber();
long double GetEpochDay();
long double GetRevSquared();
long double GetRevCubed();
long double GetBStarDrag();
long double GetSemiMajorAxis();
long double GetEccentricity();
long double GetRightAscension();
long double GetInclination();
long double GetArgumentOfPerigee();
long double GetMeanAnomaly();
long double GetEccentricAnomaly();
long double GetTrueAnomaly();
long double GetScalarRadius();
long double GetMeanMotion();
char* Satellite::GetTLELine1();
char* Satellite::GetTLELine2();

```

```

private :
    long double SemiMajorAxis;
    long double Eccentricity;
    long double RightAscension;
    long double Inclination;
    long double ArgumentOfPerigee;
    long double MeanAnomaly;
    long double EccentricAnomaly;
    long double TrueAnomaly;
    long double ScalarRadius;
    char      Name[MAXNAMELENGTH];
    long int   SSCNumber;
    long int   RevAtEpoch;
    char      SecurityClass[CLASSLENGTH+1];
    char      InternationalID[INTNUMLENGTH+1];
    int        EpochYear;
    int        EphemerisType;
    int        ElementSetNumber;
    long double EpochDay;
    long double RevSquared;
    long double RevCubed;
    long double BStarDrag;
    long double MeanMotion;
    char      TLELine1[MAXINPUTLINELENGTH];
    char      TLELine2[MAXINPUTLINELENGTH];
};

```

```

/*****
/*  THIS STRUCTURE HOLDS AN ARRAY OF SATS  */
*****/

```

```

struct SatStructure {
    Satellite Sat[MAXSATELLITES];
    int NumSats;
};

#endif

```

## B.2.3 LaserConstants.h

LaserConstants.h is the header file where all of the physical constants for the Preprocessor and the Main Processor are defined. The header file follows.

```

/*****
/* MODULE NAME:      LaserConstants.h
/* AUTHOR:          Captain David Vloedman
/* DATE CREATED:    July 27, 1998
/*
/* PURPOSE:         This module houses some of the basic constants used in
/*                  the deconfliction of a laser beam with the path of a
/*                  satellite.
/*
/* COMPILER:        Borland C++ Builder3 Standard version
/*                  This compiler should be used to compile and link.
/*
*****/
#ifndef LaserConstantsH
#define LaserConstantsH

/*****
/*****      CONSTANT DEFINITIONS      *****/
/*****

#define MAXNAMELENGTH      50          /* EACH NAME CAN BE ONLY 50 CHARS MAX */
#define GRAVITYCONSTANT    398601000 /* m/sec
#define MAXMESSAGELENGTH  300         /* MAXIMUM LENGTH OF AN ERROR MESSAGE */
#define MAXERRORS          50         /* MAX NUMBER OF ERROR MESSAGES
#define MAXSATELLITES      1000       /* MAX SATELLITES THAT CAN BE READ
#define NOERRORS           0          /* BOOLEAN ERROR FLAG
#define ERRORFOUND         1          /* BOOLEAN ERROR FLAG
#define MAXINPUTLINELENGTH 70         /* MAXIMUM CHARS OF LINE IN INPUT FILE*/
#define EARTHRAIUS        6378.135   /* EARTH RADIUS IN KILOMETERS
#define MUEARTH            398601     /* GRAV CONSTANT IN km3/sec2
#define PI                 3.14159265358979/* OBVIOUS
#define TWOPI              6.283185307179586/* OBVIOUS
#define DEGTOADIANS        0.01745329252 /* DEGREE TO RADIAN CONVERSION FACTOR */
#define RADTODEGREES       57.2957795131 /* RADIAN TO DEGREE CONVERSION FACTOR */
#define MINUTESPERDAY      1440       /* DAYS TO MINUTES CONVERSIONS FACTOR */
#define SECSSIDEREALDAY    86164.09054 /* # OF SECONDS IN A SIDEREAL DAY
#define SECSPER24HOURS     86400      /* # OF SECONDS IN 24 HOURS
#define SECSPERHOUR        3600       /* # OF SECONDS IN AN HOUR
#define LATEREERENCE       31536000   /* TIME IN SECONDS BY WHICH THETA G
/* CAN BE SAFELY PROPAGATED. NOTE:
/* 31536000 = 365*24*3600 (1 YEAR IN
/* SECONDS)
#define MMREVSPERDAY      8681660.4  /* THIS IS USED TO EXTRACT THE SEMI
/* MAJOR AXIS FROM THE MEAN MOTION
/* REVOLUTIONS PER DAY. IE:

```

```

/* MM = 8681660.4 * a^(-3/2) */
/*****
/* THE FOLLOWING CONSTANTS DEFINE BOTH THE STARTING POSITIONS */
/* OF EACH OF THE INPUT FIELDS IN THE TWO LINE ELEMENT */
/* (or TLE) INPUT FILE AND THE LENGTH OF THOSE FIELDS. THEY */
/* ARE EXPRESSED IN TERMS OF CHARACTER POSITION FROM THE */
/* BEGINNING OF THE LINE (POS), AND LENGTH OF FIELD (LENGTH). */
/* THE LENGTH IS ALSO IN CHARACTERS OF THE FIELD (NOT DIGITS). */
*****/
#define CARDPOS 1 /* CARD NUMBER */
#define CARDLENGTH 1
#define SSCPOS 3 /* SSC NUMBER */
#define SSCLENGTH 5
#define CLASSPOS 8 /* SECURITY CLASSIFICATION */
#define CLASSLENGTH 1
#define INTNUMPOS 10 /* INTERNATIONAL NUMBER */
#define INTNUMLENGTH 8
#define EYEARPOS 19 /* EPOCH YEAR */
#define EYEARLENGTH 2
#define EDAYPOS 21 /* EPOCH DAY */
#define EDAYLENGTH 12
#define REV2POS 34 /* REVOLUTIONS PER DAY SQUARED */
#define REV2LENGTH 10
#define REV3POS 45 /* REVOLUTIONS PER DAY CUBED */
#define REV3LENGTH 6
#define REVPOWERPOS 51 /* REVOLUTIONS PER DAY CUBED */
#define REVPOWERLENGTH 2
#define BSTARPOS 54 /* BSTAR DRAG */
#define BSTARLENGTH 6
#define BPOWERPOS 60 /* BSTAR DRAG */
#define BPOWERLENGTH 2
#define ETYPEPOS 63 /* EPHEMERIS TYPE */
#define ETYPELENGTH 1
#define ELSETPOS 65 /* ELEMENT SET NUMBER */
#define ELSETLENGTH 4
#define INCLINPOS 9 /* INCLINATION */
#define INCLINLENGTH 8
#define RIGHTASPOS 18 /* RIGHT ASCENSION */
#define RIGHTASLENGTH 8
#define ECCPOS 27 /* ECCENTRICITY */
#define ECCLENGTH 7
#define ARGPERPOS 35 /* ARGUMENT OF PERIGEE */
#define ARGPERLENGTH 8
#define MEANANPOS 44 /* MEAN ANOMALY */
#define MEANANLENGTH 8
#define MEANMOPOS 53 /* MEAN MOTION (Revolutions per day) */
#define MEANMOLENGTH 11
#define EPOCHREVPOS 64 /* REVOLUTION NUMBER AT EPOCH */
#define EPOCHREVLENGTH 5

#endif

```

### B.3 The Error Structure Project

This portion of the preprocessor deals with the handling, recording, and displaying of errors within the software. The error handling modules are used throughout both the Preprocessor and the Main Processor. The three modules that are used for error handling are all held in ErrorStructure.cpp. These three modules are **CreateDisplayText**, **AddError**, and **GrabError**. ErrorStructure.cpp also defines the ErrorStructure object, that is used to store all errors recorded. The Test Error Project has only a text-driven user interface that can be run in the DOS environment.

#### B.3.1 AddError

This module will add an error to the ErrorStructure, given information about the error being recorded. It receives this information via three input parameters described below.

##### Inputs

**char moduleName[MAXNAMELENGTH]** The Text name of the software module in which the error occurred. MAXNAMELENGTH is defined in LaserConstants.h.

**char description[MAXMESSAGELENGTH]** A text description of the error. MAXMESSAGELENGTH is defined in LaserConstants.h.

**int severity** The severity of the error:  
0 = Warning only  
1 = Critical Error (An error terminal to the program)

AddError gives no tangible outputs, but loads the information into the ErrorStructure.

### B.3.2 GrabError

GrabError grabs an error from the ErrorStructure. As inputs, GrabError requires only the number of the error to be retrieved.

#### Inputs

**int     number**    This is the only input GrabError requires. It is the number (between 1 and MAXERRORS) of the error to be fetched.

#### Outputs

**char   moduleName[MAXNAMELENGTH]**   The module where the error being fetched occurred.

**char   description[MAXMESSAGELENGTH]**   The description of the error being fetched.

**int     &severity**   The severity of the error:  
         0 = Warning only  
         1 = Critical Error (An error terminal to the program)

**int     &found**   A boolean flag telling whether the error asked for was found:  
         0 = Not found  
         1 = Found

### B.3.3 CreateDisplayText

There was a need to convert the errors from the ErrorStructure format to straight text, so that the errors could be accessed by outside interfaces that only recognize C-Strings. This module converts the list of errors in the ErrorStructure to an ordinary array of type char.

#### Inputs

**ErrorStructure   &errors**    This would be the variable of type ErrorStructure that holds the errors to be converted.

## Outputs

**char text [MAXERRORS] [MAXMESSAGELENGTH]** This is a two-dimensional array of characters that holds a one-line combined description of each error in the ErrorStructure.

The header file describing AddError, GrabError, and CreateDisplayText, and the rest of the ErrorStructure library, follows. Notice the modules CriticalError and WarningError are used to assess whether any errors of those respective types have occurred.

### **B.3.4 The ErrorStructure.h Header File**

```

/*****
/*  MODULE NAME:      ErrorStructure.h
/*  AUTHOR:          Captain David Vloedman
/*  DATE CREATED:    July 25, 1998
/*
/*  PURPOSE:         This module of code houses the error structure which
/*                   will be used to hold and trap any error conditions that
/*                   arise during the operation of the program.
/*
/*  COMPILER:        Borland C++ Builder3 Standard version
/*                   This compiler should be used to compile and link.
/*
*****/

#ifndef ErrorStructureH
#define ErrorStructureH

#include "LaserConstants.h"

class ErrorStructure {
public:
    ErrorStructure();          /*  CONSTRUCTOR    */
    ~ErrorStructure();         /*  DESTRUCTOR   */

/*****
/*      ErrorStructure MANIPULATION FUNCTIONS
*****/
/*****
/*  FUNCTION NAME:  AddError
/*  AUTHOR:        Captain David Vloedman
/*  DATE CREATED:   July 25, 1998
/*
/*  PURPOSE:       This function is used to record an error into the error
/*                   structure.
*****/
    void AddError(char moduleName[MAXNAMELENGTH],
                  char description[MAXMESSAGELENGTH],
                  int severity);

```

```

/*****
/* FUNCTION NAME: GrabError */
/* AUTHOR: Captain David Vloedman */
/* DATE CREATED: July 25, 1998 */
/*
/* PURPOSE: This function is used to retrieve an error that has been*/
/* previously added to the error structure. */
/*****
void GrabError(int number,
               char moduleName[MAXNAMELENGTH],
               char description[MAXMESSAGELENGTH],
               int &severity,
               int &found);

/*****
/* FUNCTION NAME: CriticalError */
/* AUTHOR: Captain David Vloedman */
/* DATE CREATED: July 25, 1998 */
/*
/* PURPOSE: This function is used to determine if a critical (fatal)*/
/* error has been detected and recorded yet. */
/* CriticalErrorFound = 1 --> TRUE */
/* CriticalErrorFound = 0 --> FALSE */
/*
/*****
int CriticalError();

/*****
/* FUNCTION NAME: WarningError */
/* AUTHOR: Captain David Vloedman */
/* DATE CREATED: July 25, 1998 */
/*
/* PURPOSE: This function is used to determine if a warning (non-*/
/* fatal) error has been detected and recorded yet. */
/* WarningFound = 1 --> TRUE */
/* WarningFound = 0 --> FALSE */
/*
/*****
int WarningError();

/*****
/* FUNCTION NAME: TotalErrors */
/* AUTHOR: Captain David Vloedman */
/* DATE CREATED: July 25, 1998 */
/*
/* PURPOSE: This function is used to determine how many errors total*/
/* have occurred and been recorded. */
/* ErrorsFound = Total number of errors. */
/*
/*****
int TotalErrors();

/*****
/* These private structures cannot be seen */
/* outside this module. They are used to */
/* errors and are interfaced with by the */
/* public functions. */

```

```

/*****
private :
    int CriticalErrorFound;
    int WarningFound;
    int ErrorsFound;
    char ModuleList[MAXERRORS][MAXNAMELENGTH];
    char ErrorList[MAXERRORS][MAXMESSAGELENGTH];
    int Severity[MAXERRORS];
};

/*****
/* FUNCTION NAME: CreateDisplayText */
/* AUTHOR: Captain David Vloedman */
/* DATE CREATED: July 25, 1998 */
/*
/* PURPOSE: This function is used to create a simple array of */
/* character arrays which hold all of the information */
/* held in the error-structure. This two-dimensional */
/* text array may have messages as long as MAXMESSAGELENGTH*/
/* and can hold MAXERRORS messages. */
/*
/* INPUTS: NAME: PURPOSE: */
/* errors DataStructure holding all errors */
/*
/* OUTPUTS: NAME: PURPOSE: */
/* text A completely textual version of */
/* errors. */
/*****/
void CreateDisplayText(ErrorStructure &errors,
    char text[MAXERRORS][MAXMESSAGELENGTH]);
#endif

```



## B.4 The SGP4 Support Library

The SGP4 Support Library consists of all of the modules used to store and interface with the SGP4 satellite ephemeris propagator. Although SGP4 was written independently of this project by Air Force Space Command, a copy of it (version 3.01C) is stored in the module SGP4Routines.cpp. These external routines are accessed using the module "CallSGP4", stored in the SGP4SupportModules library. CallSGP4, and consequently SGP4 itself, can be tested using the Graphical User Interface illustrated in Figure B.3. The GUI below is controlled by the Cbuilder module SGP4TestForm, which was developed to facilitate testing of the nature and behavior of the interface to SGP4.

SATELLITE INFORMATION		TESTING CRITERIA	
SSC Number	0	X (km)	0
Classification	0	Y (km)	0
International ID	0	Z (km)	0
Epoch Year	1998	Xdot (km/sec)	0
Epoch Day	216.91470564	Ydot (km/sec)	0
Revs /Day Squared	0	Zdot (km/sec)	0
Revs /Day Cubed	0	Delta Time (Min)	0
BStar Drag	0	Inclination	0
Ephemeris Type	0	Right Ascension	0
		Eccentricity	0
		Mean Motion	0
		Arg of Perigee	0
		Mean Anomaly	0

Populate Using File: PAData/LEOFile1.txt

Run SGP4 (Version 3.01)

Error Messages: No Errors

**Figure B.3. Testing GUI Used to Access CallSGP4**

### B.4.1 CallSGP4

As mentioned previously, CallSGP4 is the module used to call and interface with SGP4. The input and output interface used by SGP4 is proprietary to the Air Force, and will not be discussed here, however the interface to CallSGP4 can be explained.

#### Inputs

**struct Satellite &Sat** This first input is just the Satellite object that holds all of the ephemeris information gleaned from a Two-Line Element Set file, and populated using the ReadTLEFile module.

**double JulianDate** This is the modified Julian Date (The Julian Date – 2440000) that needs to be propagated to. This is the actual time at which the user wishes to find the position of the satellite.

#### Outputs

**double &X** The X coordinate of the satellite at the Julian Date specified, given in terms of the ECI frame, in kilometers.

**double &Y** The Y coordinate of the satellite at the Julian Date specified, given in terms of the ECI frame, in kilometers.

**double &Z** The X coordinate of the satellite at the Julian Date specified, given in terms of the ECI frame, in kilometers.

**double &Xdot** The velocity in the X direction (ECI frame) of the satellite at the Julian Date specified, in kilometers per second.

**double &Ydot** The velocity in the Y direction (ECI frame) of the satellite at the Julian Date specified, in kilometers per second.

**double &Zdot** The velocity in the Z direction (ECI frame) of the satellite at the Julian Date specified, in kilometers per second.

**double &Inclination** The inclination of the satellite at the Julian Date specified, in degrees.

**double &RightAscension** The Right Ascension of the satellite at the Julian Date specified, in degrees.

**double &Eccentricity** The Eccentricity of the satellite at the Julian Date specified, in degrees.

**double &MeanMotion** The Mean Motion of the satellite at the Julian Date specified.

**double &ArgumentOfPerigee** The Argument of Perigee of the satellite at the Julian Date specified, in degrees.

**double &MeanAnomaly** The Mean Anomaly of the satellite at the Julian Date specified, in degrees.

**double &Delta** This is the time that has elapsed between the time that the original ephemeris data for the satellite (held in the Satellite object) and the Julian propagation date specified. In other words, the amount of time (in minutes) that has been propagated.

**ErrorStructure &ErrorList** The error handling object.

#### B.4.2 The SGP4SupportModules.h Header File

```

/*****
/*  MODULE NAME:      SGP4SupportModules.h
/*  AUTHOR:          Captain David Vloedman
/*  DATE CREATED:    October 20, 1998
/*
/*  PURPOSE:         This set of modules supports incorporating "SGP4", a
/*                   Satellite position/time propagator developed by
/*                   United States Space Command. These modules were
/*                   developed for SGP4 Version 3.01C. They simply serve as
/*                   an interface between this project and SGP4.
/*
/*  COMPILER:        Borland C++ Builder3 Standard version
/*                   This compiler should be used to compile and link.
/*
*****/
#ifndef SGP4SupportModulesH
#define SGP4SupportModulesH

#include "ErrorStructure.h"
/*****
/*                   FUCTIONS
*****/

/*****
/*  FUNCTION NAME:    CallSGP4
/*  AUTHOR:          Captain David Vloedman
/*  DATE CREATED:    October 20, 1998
/*
/*  PURPOSE:         This procedure will take elements already existing

```

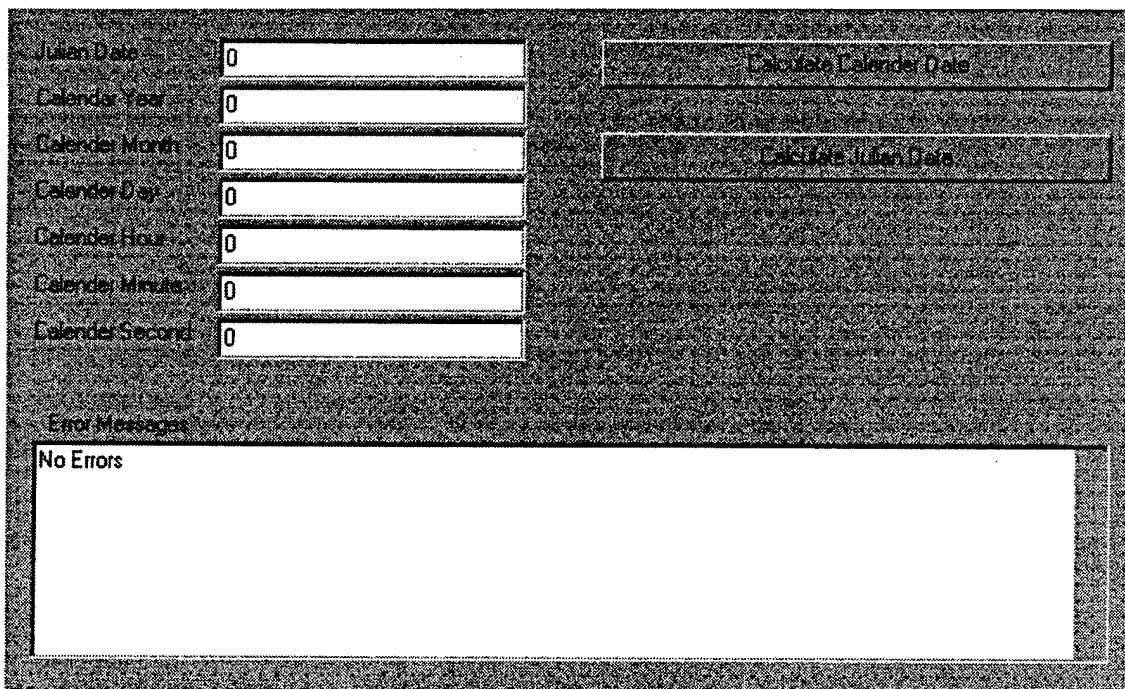
```

/*          within the Predictive Avoidance Project code and adapt */
/*          that information slightly to be used by SGP4 version */
/*          3.01. It will then make a call to SGP4 and return the */
/*          results. */
/*          */
/* INPUTS:      NAME:      DEFINITION: */
/*          Sat          Holds all ephemeris information */
/*                      for the Satellite being studied */
/*          JulianDate    The time to which the position */
/*                      of sat should be propagated to */
/* OUTPUTS:      NAME:      DESCRIPTION: */
/*          X            X axis pos in ECI frame at Jul */
/*                      date */
/*          Y            Y axis pos in ECI frame at Jul */
/*                      date */
/*          Z            Z axis pos in ECI frame at Jul */
/*                      date */
/*          Xdot          Velocity vector in X direction */
/*          Ydot          Velocity vector in Y direction */
/*          Zdot          Velocity vector in Z direction */
/*          Inclination   Inclination at Julian Date */
/*          RightAscension Right Ascension at Julian Date */
/*          Eccentricity  Eccentricity at Julian Date */
/*          ArgumentOfPerigee Arg of Perigee at Julian Date */
/*          Mean Anomaly  The Mean Anomaly at Julian Date */
/*          Delta          The amount of time in seconds */
/*                      that has transpired between the */
/*                      actual ephemeris measurements */
/*                      and the Julian Date propagated */
/*          ErrorList      The Errors which have occurred */
/*          */
/*          */
/* COMPILER:      Borland C++ Builder3 Standard version */
/*          This compiler should be used to compile and link. */
/*          */
/*****
CallSGP4(struct Satellite &Sat,
        double JulianDate,
        double &X,
        double &Y,
        double &Z,
        double &Xdot,
        double &Ydot,
        double &Zdot,
        double &Inclination,
        double &RightAscension,
        double &Eccentricity,
        double &MeanMotion,
        double &ArgumentOfPerigee,
        double &MeanAnomaly,
        double &Delta,
        ErrorStructure &ErrorList);

```

## B.5 The Time Module Library

The Time Module Test project consists of two modules that convert back and forth between the Calendar Date and the Modified Julian Date. These two modules are **ConvertCalendarToJulian** and **ConvertJulianToCalendar**. They are both stored in the TimeModule.cpp library. Both of these modules can be tested independently with any calling routine. The graphical interface shown in Figure B.4 has been developed for this purpose.

The image shows a graphical user interface (GUI) for testing time modules. It features a dark background with several input fields and buttons. On the left, there are seven input fields labeled 'Julian Date', 'Calendar Year', 'Calendar Month', 'Calendar Day', 'Calendar Hour', 'Calendar Minute', and 'Calendar Second'. Each field contains the number '0'. To the right of these fields are two buttons: 'Calculate Calendar Date' and 'Calculate Julian Date'. Below the input fields is a section labeled 'Error Messages' containing a large white rectangular area with the text 'No Errors'.

**Figure B.4.** Graphical Interface Developed for Testing the Time Modules

The code for this GUI is contained within the C++ Builder module TimeTestForm, and is included in *Appendix E*.

### **B.5.1 ConvertCalenderToJulian**

The module, ConvertCalendarToJulian will take a date in the modern calendar, down to a fraction of a second, and convert it to its equivalent Modified Julian Date. The Modified Julian Date is simply the Julian Date – 2440000 days.

#### **Inputs**

**int Cyear** The Calendar Year (all four digits) of the date to be converted to the Modified Julian Date.

**int Cmonth** The Calendar Month (1 to 12) of the date to be converted to the Modified Julian Date.

**int Cday** The Calendar Day (1 to 366) of the date to be converted to the Modified Julian Date.

**int Chour** The Calendar Hour (0 to 24) of the date to be converted to the Modified Julian Date.

**int Cminute** The Calendar Minute (0 to 60) of the date to be converted to the Modified Julian Date.

**double Csecond** The Calendar Second (0 – 59.99999999) of the date to be converted to the Modified Julian Date.

#### **Outputs**

**double &JulianDate** The Modified Julian Date converted from the Calendar Date above.

**ErrorStructure &ErrorList** The error-handling structure.

### **B.5.2 ConvertJulianToCalendar**

The ConvertJulianToCalendar module does just the reverse of its sister module. It will take a Modified Julian Date and convert it to its equivalent calendar date.

## Inputs

**double JulianDate** The Modified Julian to be converted to an equivalent Calendar Date.

## Outputs

**int &Cyear** The Calendar Year (all four digits) of the date converted from the Modified Julian Date.

**int &Cmonth** The Calendar Month (1 to 12) of the date converted from the Modified Julian Date.

**int &Cday** The Calendar Day (1 to 366) of the date converted from the Modified Julian Date.

**int &Chour** The Calendar Hour (0 to 24) of the date converted from the Modified Julian Date.

**int &Cminute** The Calendar Minute (0 to 60) of the date converted from the Modified Julian Date.

**double &Csecond** The Calendar Second (0 – 59.99999999) of the date converted from the Modified Julian Date.

**ErrorStructure &ErrorList** The error-handling structure.

### B.5.3 The TimeModule.h Header File

```

/*****
/*  MODULE NAME:      TimeModules.h
/*  AUTHOR:          Captain David Vloedman
/*  DATE CREATED:     September 10, 1998
/*
/*  PURPOSE:         This module of code houses the Time routines which are
/*                   used to retrieve and manipulate the times used as
/*                   reference times for satellite passing. The numerical
/*                   algorithms were provided by Professor William Wiesel,
/*                   Air Force Institute of Technology, who earlier gleaned
/*                   the algorithms from the text, "Numerical Recipes". It
/*                   was converted from Fortran to C++ by the author.
/*
/*  COMPILER:         Borland C++ Builder3 Standard version
/*                   This compiler should be used to compile and link.
/*
*****/
#ifndef TimeModulesH
#define TimeModulesH
/*****/

```

```

/* USER-BUILT LIBRARIES */
/*****
#include "ErrorStructure.h"
/*****
/***** FUNCTIONS *****/
/*****

/* FUNCTION NAME: ConvertCalenderToJulian */
/* AUTHOR: Captain David Vloedman */
/* DATE CREATED: September 10, 1998 */
/*
/* PURPOSE: This function will read in the calender date and return */
/* the equivalent modified Julian date. */
/*
/* INPUTS: NAME: DEFINITION: */
/* CYear Holds the calender year */
/* Cmonth Holds the Calender month(1 - 12) */
/* CDay Holds calender day */
/* CHour Holds the calender hour */
/* CMinute Holds the calender minute */
/* CSecond Holds the calender second */
/* ErrorList Holds the Errors found */
/*
/* OUTPUTS: NAME: DEFINITION: */
/* JulianDate Holds the Julian equivalent to */
/* the calender date. */
/*
/* COMPILER: Borland C++ Builder3 Standard version */
/* This compiler should be used to compile and link. */
/*
/*****
void ConvertCalenderToJulian(int CYear,
                           int CMonth,
                           int CDay,
                           int CHour,
                           int CMinute,
                           double Csecond,
                           double &JulianDate,
                           ErrorStructure &ErrorList);

/*****
/* FUNCTION NAME: ConvertJulianToCalender */
/* AUTHOR: Captain David Vloedman */
/* DATE CREATED: September 10, 1998 */
/*
/* PURPOSE: This function will read in the Julian date and return */
/* the equivalent calender date. */
/*
/* INPUTS: NAME: DEFINITION: */
/* JulianDate Holds the Julian equivalent to */
/* the calender date. */
/*
/* OUTPUTS: NAME: DEFINITION: */
/* CYear Holds the calender year */
/* Cmonth Holds the Calender month(1 - 12) */
/* CDay Holds calender day */
/* CHour Holds the calender hour */
/* CMinute Holds the calender minute */
/* CSecond Holds the calender second */
/* ErrorList Holds the Errors found */

```



```

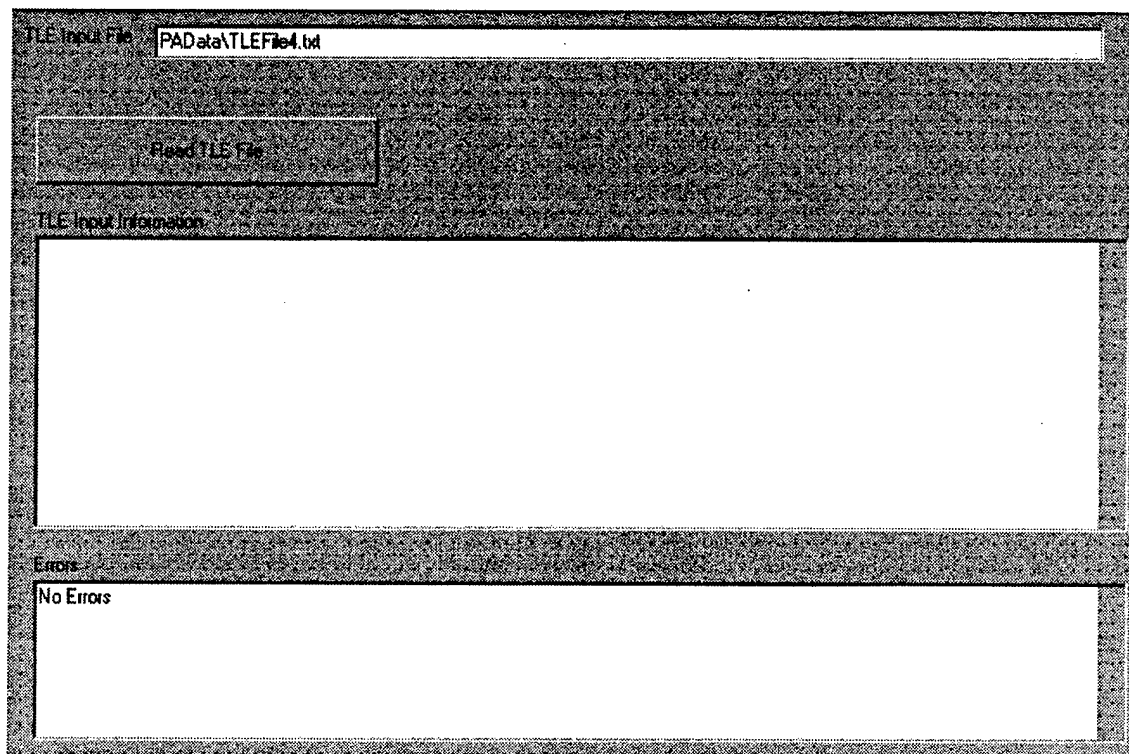
/*                                                                    */
/* COMPILER:      Borland C++ Builder3 Standard version              */
/* This compiler should be used to compile and link.                 */
/*                                                                    */
/*****
void ConvertJulianToCalender(int &CYear,
                             int &CMonth,
                             int &CDay,
                             int &CHour,
                             int &CMinute,
                             double &CSecond,
                             double JulianDate,
                             ErrorStructure &ErrorList);

#endif

```

## B.6 The TLE Input Library

The TLE Input Library consists of the interface module used to read all of the input that comes to the software package in the form of Two-Line Element (TLE) set files. The TLE data format is used widely to hold satellite ephemeris in a data file. It is by the popular software package, Satellite Tool Kit, developed by Analytical Graphics to hold satellite information, as well as by Air Force Space Command and a host of other users. This is the most likely format of the satellite ephemeris data that must inevitably be downloaded to the Preprocessor (and Main processor) for analysis. The Module, ReadTLEFile, is the module responsible for reading this type of formatted input file and loading the information into an object of type SatStructure which is defined in the



**Figure B.5.** Graphical Interface Developed for Testing ReadTLEFile

Satellite.h module. ReadTLEFile is housed in the TLEInput.cpp library. ReadTLEFile can be called from any C++ program, and it can be tested using the Graphical C++Builder module, TLETestForm, which generates the GUI illustrated in Figure B.5.

### B.6.1 ReadTLEFile

The ReadTLEFile module is the module that reads a TLE file and populates SatStructure with the satellite data contained inside of it. The format of a sample TLE file is shown in *Appendix F*.

#### Inputs

**char FileName[MAXNAMELENGTH]** The only input the ReadTLEFile is the name of the TLE file to be read.

#### Outputs

**struct SatStructure &SatArray** SatArray is an object of type SatStructure, which is essentially an array of Satellite objects. It is defined in the Satellite.h file.

**ErrorStructure &ErrorList** The error-handling structure.

### B.6.2 The TLE Input.h Header File

```

/*****
/*  MODULE NAME:      TLEInput.h
/*  AUTHOR:          Captain David Vloedman
/*  DATE CREATED:    August 18, 1998
/*
/*  PURPOSE:         This module of code houses the routines which input the
/*                   Two Line Element (TLE) sets from an input file.
/*
/*  COMPILER:        Borland C++ Builder3 Standard version
/*                   This compiler should be used to compile and link.
/*
/*****
#ifndef TLEInputH
#define TLEInputH
#include "LaserConstants.h"
#include "Satellite.h"
#include "ErrorStructure.h"
/*****
/*****          FUCTIONS          *****/

```

```

/*****
/*****
/* FUNCTION NAME:  ReadTLEFile                               */
/* AUTHOR:        Captain David Vloedman                     */
/* DATE CREATED:   August 18, 1998                           */
/*                                                        */
/* PURPOSE:       This function will read in the information contained in */
/*                an input file holding Two Line Element (TLE) sets.      */
/*                These TLEs hold the ephemeris data for all of the        */
/*                satellites we will be covering. It uses the TLE          */
/*                information to populate a satellite data structure which*/
/*                is used throughout the program.                  */
/*                                                        */
/* INPUTS:        NAME:            DEFINITION:                */
/*                FileName          Holds the name of the Input File*/
/*                                                        */
/* OUTPUTS:       NAME:            DEFINITION:                */
/*                SatArray          Returns satellite information    */
/*                ErrorList         Returns error information        */
/*                                                        */
/* COMPILER:      Borland C++ Builder3 Standard version          */
/*                This compiler should be used to compile and link.    */
/*                                                        */
/*****
void ReadTLEFile(char FileName[MAXNAMELENGTH],
                 struct SatStructure &SatArray,
                 ErrorStructure &ErrorList);
#endif

```

## B.7 The Evaluate Ephemeris Library

The purpose of the Evaluate Ephemeris portion of the preprocessor is to tie together all of the other modules and analyze the data for just one satellite. This library is, therefore, the heart of the preprocessor. It can be used to more intensely scrutinize a single satellite engagement for error checking or other purposes. The library contains three modules, **EvaluateEphemeris**, **CompareOrbit**, and **FindThetaG**. Each module can be run independently as a stand alone application, and all are run repeatedly by each execution of the preprocessor. A Graphical Interface has been created using C++Builder 3 to execute these three modules using a single satellite's ephemeris input. This interface is shown in Figure B.6, and is controlled by the module, EvaluateEphemerisForm. The implementation for this module is contained in *Appendix E*.

Satellite Information		Position Information		Velocity Information		TESTING CATEGORIES	
Satellite Name	0 S	Position Number	0	Position X (km)	0	Satellite X (km)	0
Position Latitude	0	Classification	0	Position Y (km)	0	Satellite Y (km)	0
Position Longitude	0	International ID	0	Position Z (km)	0	Satellite Z (km)	0
Position Latitude Degrees	60	Epoch Year	0	Position X Velocity (km/s)	0	Satellite X Velocity (km/s)	0
Position Latitude Minutes	0	Epoch Day	0	Position Y Velocity (km/s)	0	Satellite Y Velocity (km/s)	0
Position Latitude Seconds	0	Time Day Epoch	0	Position Z Velocity (km/s)	0	Satellite Z Velocity (km/s)	0
Position Altitude (km)	12.9	Time Day Epoch	0	Position X Acceleration (km/s²)	0	Satellite X Acceleration (km/s²)	0
Position X (km)	300	Time Day Epoch	0	Position Y Acceleration (km/s²)	0	Satellite Y Acceleration (km/s²)	0
Position Y (km)	300	Time Day Epoch	0	Position Z Acceleration (km/s²)	0	Satellite Z Acceleration (km/s²)	0
Position Z (km)	0	Time Day Epoch	0	Position X Jerk (km/s³)	0	Satellite X Jerk (km/s³)	0
Greenwich Time Reference		Greenwich Mean Sidereal Time		Theta D in Degrees		Satellite X Jerk (km/s³)	
Calculation Year	1998	Reference Hour	4	Theta D in Degrees	0	Satellite Y Jerk (km/s³)	0
Calculation Month (1-12)	8	Reference Minute	40	Satellite in Range?	NO	Satellite Z Jerk (km/s³)	0
Calculation Day	08	Reference Second	1.299	Ephemeris in Range?	NO	Satellite X Jerk (km/s³)	0
Calculation Hour	2	Modified Julian Date	11029.0	Ephemeris in Range?	NO	Satellite Y Jerk (km/s³)	0
Calculation Minute	58	Seconds To Next Run	5	Evaluate Ephemeris View		Satellite Z Jerk (km/s³)	0
Calculation Second	2.0					Satellite X Jerk (km/s³)	0
No Errors							

Figure B.6. Graphical Interface Used to Test EvaluateEphemeris

### B.7.1 EvaluateEphemeris

EvalaluateEphemeris is the module that calls all of the modules so far discussed, including CompareOrbit and FindThetaG. It is the pinnacle module responsible for tying together all of the data and algorithms together for a single satellite analysis. It is called multiple times by the preprocessor to analyze each satellite in the input file in succession. This module is responsible for determining whether or not a satellite is or will be in the field of view of the platform during a given time increment.

#### Inputs

**struct Satellite &Sat** This first input is just the Satellite object that holds all of the ephemeris information gleaned from a Two-Line Element Set file, and populated using the ReadTLEFile module.

**struct Aircraft &ABLPlatform** ABLPlatform is a structure of type "Aircraft" that holds all of the information about the position of the aircraft at the time of execution of the Preprocessor.

**double ThetaGInRad** This is the angle at which the Earth's Greenwich Meridian is currently at with respect to the ECI frame, where the referent angle is the Vernal Equinox. This angle should be in radians.

**double TimeToNextRun** The estimated time until the next run of the Preprocessor.

**double JulianDate** This is the modified Julian Date (The Julian Date - 2440000) that needs to be propagated to. This is the actual time at which the user wishes to find the position of the satellite.

#### Outputs

**int &SatelliteInView** This is a boolean variable that tells whether or not the satellite being evaluated is currently (as of the Julian Date given) in view of the platform given:

1 = satellite in view

0 = satellite not in view

**int        &OrbitInView**    It may be that the satellite is not in view, but its ephemeris, or the path the satellite follows, is currently in view. This is regardless of whether the satellite itself is in view. Naturally, if the satellite is in view, the orbit must also be in view.

1 = orbit in view

0 = orbit not in view

**double    &SatX**    The X coordinate of the satellite at the Julian Date specified, given in terms of the ECI frame, in kilometers.

**double    &SatY**    The Y coordinate of the satellite at the Julian Date specified, given in terms of the ECI frame, in kilometers.

**double    &SatZ**    The X coordinate of the satellite at the Julian Date specified, given in terms of the ECI frame, in kilometers.

**double    &SatXdot**    The velocity in the X direction (ECI frame) of the satellite at the Julian Date specified, in kilometers per second.

**double    &SatYdot**    The velocity in the Y direction (ECI frame) of the satellite at the Julian Date specified, in kilometers per second.

**double    &SatZdot**    The velocity in the Z direction (ECI frame) of the satellite at the Julian Date specified, in kilometers per second.

**double    &Delta**    This is the time that has elapsed between the time that the original ephemeris data for the satellite (held in the Satellite object) and the Julian propagation date specified. In other words, the amount of time (in minutes) that has been propagated.

**double    &Inclination**    The inclination of the satellite at the Julian Date specified, in degrees.

**double    &RightAscension**    The Right Ascension of the satellite at the Julian Date specified, in degrees.

**double    &Eccentricity**    The Eccentricity of the satellite at the Julian Date specified, in degrees.

**double    &MeanMotion**    The Mean Motion of the satellite at the Julian Date specified.

**double    &ArgumentOfPerigee**    The Argument of Perigee of the satellite at the Julian Date specified, in degrees.

**double &MeanAnomaly** The Mean Anomaly of the satellite at the Julian Date specified, in degrees.

**double &Dvector** The Dvector is the vector used to evaluate the time to rise. Its presence here is used mostly for testing purposes and can be largely ignored. For a more complete explanation, see Chapter 2, pages 18-31.

**double &TimeToRise** If the orbit of the satellite is in view, but the satellite is not in view, this parameter gives the time estimate of when the satellite is expected to come into view.

**double &CriticalRadius** The Critical Radius describes the smallest radius at the satellites position that can appear above the artificial horizon of the platform. This parameter is also used mostly for testing, and can be disregarded when called by other applications. For a more complete explanation, see Chapter 2, pages 18-31.

**double &SatRadius** The Sat Radius describes the radius of the satellite as measured from the center of the Earth. This parameter is also used mostly for testing, and can be disregarded when called by other applications. For a more complete explanation, see Chapter 2, pages 18-31.

**ErrorStructure &ErrorList** The error handling object.

### B.7.2 CompareOrbit

Compare Orbit is the module used by EvaluateEphemeris to see if the orbit of the satellite is in view of the platform.

#### Inputs

**struct Satellite &Sat** This first input is just the Satellite object that holds all of the ephemeris information gleaned from a Two-Line Element Set file, and populated using the ReadTLEFile module.

**struct Aircraft &ABLPlatform** ABLPlatform is a structure of type "Aircraft" that holds all of the information about the position of the aircraft at the time of execution of the Preprocessor.

**double ThetaGInRad** This is the angle at which the Earth's Greenwich Meridian is currently at with respect to the ECI frame, where the referent angle is the Vernal Equinox. This angle should be in radians.



## Outputs

**double &TimeToRise** If the orbit of the satellite is in view, but the satellite is not in view, this parameter gives the time estimate of when the satellite is expected to come into view.

**double &CriticalRadius** The Critical Radius describes the smallest radius at the satellites position that can appear above the artificial horizon of the platform. This parameter is also used mostly for testing, and can be disregarded when called by other applications. For a more complete explanation, see Chapter 2, pages 18-31.

**double &SatRadius** The Sat Radius describes the radius of the satellite as measured from the center of the Earth. This parameter is also used mostly for testing, and can be disregarded when called by other applications. For a more complete explanation, see Chapter 2, pages 18-31.

**ErrorStructure    &ErrorList** The error handling object.

### **B.7.3 FindThetaG**

The module, FindThetaG, is used to propagate the Earth's rotation in the ECI coordinate frame over time. It requires a reference position for the Greenwich Meridian, at a given reference time, and the Modified Julian Date of the time that is to be propagated to. It is important to remember, that, when using this module, the reference time and the propagation date should not be more than a year apart. If they are more than a year apart, the user takes the chance that accuracy will fade, making the angle less precise.

## Inputs

**int        ReferenceHour** Reference hour Refers to the Reference angle of  $\theta_g$  (The angle between the Greenwich meridian and the Vernal Equinox). This angles is given in hours, minutes and seconds as opposed to degrees or radians. This parameter holds the hours portion of  $\theta_g$ .

**int        ReferenceMinute** The minutes portion of  $\theta_g$ .

**double    ReferenceSecond** The seconds portion of  $\theta_g$ .

**double    RefModJulianDate** This parameter holds the Modified Julian Date at which the reference angle,  $\theta_g$ , was taken. This allows  $\theta_g$  to be propagated forward to the present moment.

**int        CalcYear** The current year.

**int        CalcMonth** The current month (1-12).

**int        CalcDay** The current day (1-31).

**int        CalcHour** The current hour (1-24).

**int        CalcMinute** The current minute (1-60).

**double    CalcSecond** The current second. This is the only part of the current time that can be given as a non-integer. This field should be accurate to at least three decimal places.

## Outputs

**double    &ThetaGInRadians** This is the instantaneous angle between the Greenwich meridian and the Vernal Equinox at the moment of execution of the preprocessor.

**ErrorStructure    &ErrorList)** This parameter is both an input and output parameter. Each module uses it to assess whether a fatal error has occurred somewhere else in the program, and uses it to record errors which may be important to the user.

## B.7.4 The EvaluateEphemerisModules.h Header File

```

/*****
/*  MODULE NAME:    EvaluateEphemerisModules.h
/*  AUTHOR:        Captain David Vloedman
/*  DATE CREATED:   August 18, 1998
/*
/*  PURPOSE:       This set of modules supports the preprocessor and are
/*                  used to evaluate whether or not the satellite is ever
/*                  above the platform horizon.
/*
/*  COMPILER:      Borland C++ Builder3 Standard version
/*                  This compiler should be used to compile and link.
/*
*****/
#ifndef EvaluateEphemerisModulesH
#define EvaluateEphemerisModulesH

#include "ErrorStructure.h"
#include "Aircraft.h"
#include "Satellite.h"

/*****
*****          FUCTIONS          *****/
/*****

/*****
/*  FUNCTION NAME:  EvaluateEphemeris
/*  AUTHOR:        Captain David Vloedman
/*  DATE CREATED:   Sept 19, 1998
/*
/*  PURPOSE:       This function will take the position of the aircraft and
/*                  the orbital elements of the satellite and calculate
/*                  whether or not the satellite ever comes into view (or
/*                  above the horizontal horizon) of the the aircraft.
/*
/*  INPUTS:        NAME:      DEFINITION:
/*                  Sat       Holds all ephemeris information
/*                               for the Satellite being studied
/*                  ABLPlatform Holds all information about ABL
/*                               Platform position/disposition
/*                  JulianDate The time to which the position
/*                               of sat should be propagated to
/*                  TimeToNextRun The amount of time for which the
/*                               current run must last. This is
/*                               To determine how much time in
/*                               seconds will transpire before
/*                               next update is received.
/*                  ThetaGInRadians The angle between the Greenwich
/*                               Meridian and the Vernal Equinox
/*                               at JulianDate.
/*
/*  OUTPUTS:       NAME:      DESCRIPTION:
/*                  SatelliteInView If the Satellite is visible to
/*                               the ABLPlatform (over the
/*                               artificial horizon of the
/*                               aircraft. 1 = "yes", 0 = "no"
/*                  OrbitInView   Is the satellite ever above the
/*                               horizon plain of the platform?
/*                               (IE, is the orbit itself, regard
/*                               less of the satellite present

```

```

/* position, it view? YES=1, NO=0. */
/* SatX X axis pos in ECI frame at Jul */
/* date */
/* SatY Y axis pos in ECI frame at Jul */
/* date */
/* SatZ Z axis pos in ECI frame at Jul */
/* date */
/* SatXdot Velocity vector in X direction */
/* SatYdot Velocity vector in Y direction */
/* SatZdot Velocity vector in Z direction */
/* Inclination Inclination at Julian Date */
/* RightAscension Right Ascension at Julian Date */
/* Eccentricity Eccentricity at Julian Date */
/* ArgumentOfPerigee Arg of Perigee at Julian Date */
/* Mean Anomaly The Mean Anomaly at Julian Date */
/* Delta The amount of time in seconds */
/* that has transpired between the */
/* actual ephemeris measurements */
/* and the Julian Date propagated */
/* Dvector This is the magnitude of the */
/* satellite radius vector (the */
/* vector from earth center to the */
/* satellite) in the direction of */
/* the Platform radius vector. IE */
/* the component of the sat radius */
/* vector in the Platform radius */
/* direction. This is used to show */
/* how close the sat is to rising */
/* above the artificial horizon. */
/* TimeToRise Estimated time before the sat */
/* rises above the platform's */
/* artificial horizon. */
/* CriticalRadius The Radial component which tells */
/* the minimum distance an object */
/* must be before it lies above the */
/* artificial horizon of the */
/* platform. */
/* SatRadius The Radial altitude of the sat */
/* wrt the platform altitude. This */
/* is compared to the critical rad */
/* to determine if the sat lies */
/* above or below the platform */
/* artificial horizon. */
/* ErrorList The Errors which have occurred */
/* */
/* COMPILER: Borland C++ Builder3 Standard version */
/* This compiler should be used to compile and link. */
/* */
/*****
void EvaluateEphemeris( struct Satellite &Sat,
                      struct Aircraft &Platform,
                      double ThetaGInRad,
                      double JulianDate,
                      double TimeToNextRun,
                      int &SatelliteInView,
                      int &OrbitInView,
                      double &SatX,
                      double &SatY,
                      double &SatZ,
                      double &SatXdot,
                      double &SatYdot,
                      double &SatZdot,

```

```

        double &Delta,
        double &Inclination,
        double &RightAscension,
        double &Eccentricity,
        double &MeanMotion,
        double &ArgumentOfPerigee,
        double &MeanAnomaly,
        double &Dvector,
        double &TimeToRise,
        double &CriticalRadius,
        double &SatRadius,
        ErrorStructure &ErrorList);

/*****
/*  FUNCTION NAME:  FindThetaG
/*  AUTHOR:        Captain David Vloedman
/*  DATE CREATED:   October 6, 1998
/*
/*  PURPOSE:       This function will take a reference position and time
/*                  for a known angle between the Greenwich Meridian and
/*                  the Vernal Equinox, and propagate the angle through
/*                  natural orbit precession at the given calculation time.
/*                  Note that the reference time must always be BEFORE the
/*                  calculation time.
/*
/*  INPUTS:        NAME:          DEFINITION:
/*                  ReferenceHour  This holds the value of Theta G
/*                                  at RefModJulianDate. The angle
/*                                  of Theta G is given in hours,
/*                                  minutes, and seconds instead of
/*                                  degrees, where 24 hrs = 360 deg
/*                  ReferenceMinute Holds the minutes of Theta G at
/*                                  RefModJulianDate.
/*                  ReferenceSecond Holds the seconds of Theta G at
/*                                  RefModJulianDate.
/*                  RefModJulianDate This is the reference date when
/*                                  an actual observation of the
/*                                  true value of theta G was made.
/*                  CalcYear        Holds the current calendar year
/*                  Calcmonth       Holds the Calendar month(1 - 12)
/*                  CalcDay         Holds calendar day
/*                  CalcHour        Holds the calendar hour
/*                  CalcMinute      Holds the calendar minute
/*                  CalcSecond      Holds the calendar second
/*
/*  OUTPUTS:       NAME:          DESCRIPTION:
/*                  ThetaGInRadians The angle between the Greenwich
/*                                  Meridian and the Vernal Equinox
/*                                  at Calc Date.
/*                  ErrorList       The Errors which have occurred
/*
/*  COMPILER:      Borland C++ Builder3 Standard version
/*                  This compiler should be used to compile and link.
/*
*****/
void FindThetaG(int    ReferenceHour,
                int     ReferenceMinute,
                double  ReferenceSecond,
                double  RefModJulianDate,
                int     CalcYear,
                int     CalcMonth,

```

```

        int      CalcDay,
        int      CalcHour,
        int      CalcMinute,
        double    CalcSecond,
        double    &ThetaGInRadians,
        ErrorStructure &ErrorList);

/*****
/*  FUNCTION NAME:  CompareOrbit
/*  AUTHOR:        Captain David Vloedman
/*  DATE CREATED:   October 6, 1998
/*
/*  PURPOSE:       This function will take the position of the aircraft and
/*                  the orbital elements of the satellite and calculate
/*                  whether or not the satellite ever comes into view (or
/*                  above the horizontal horizon) of the the aircraft. Note
/*                  that this is at an instantaneous time. It does not
/*                  account for the precession of the orbit, and so must
/*                  be run at regular close (30 minute) intervals to be
/*                  reliable and accurate.
/*
/*  INPUTS:         NAME:                DEFINITION:
/*      Platform.LatitudeDegree          Degree of Latitude (0-90 int)
/*      Platform.LatitudeMinute          Minute of Latitude (0-60 int)
/*      Platform.LatitudeSecond          Second of Latitude (0-60 float)
/*      Platform.LongitudeDegree         Degree of Longitude (0-360 int)
/*      Platform.LongitudeMinute         Minute of Longitude (0-60 int)
/*      Platform.LongitudeSecond         Second of Longitude (0-60 float)
/*      Sat.RightAscension               Right Ascension (degrees)
/*      Sat.Eccentricity                 Eccentricity (float)
/*      Sat.Inclination                  Inclination (degrees)
/*      Sat.MeanMotion                  Mean Motion (float)
/*      Sat.ArgumentOfPerigee            Degrees (0-360)
/*      Sat.MeanAnomaly                  Degrees (0-360)
/*      Sat.EpochDay                     Day of year msrmts taken (float)
/*      Sat.EpochYear                    Calender Year (int)
/*      ThetaGInRad                      Angle between Greenwich and
/*                                      Vernal Equinox
/*      ErrorList                        Errors that have occured
/*
/*  OUTPUTS:        NAME:                DESCRIPTION:
/*      CriticalRadius                   The Radial component which tells
/*                                      the minimum distance an object
/*                                      must be before it lies above the
/*                                      artificial horizon of the
/*                                      platform.
/*      SatRadius                        The Radial altitude of the sat
/*                                      wrt the platform altitude. This
/*                                      is compared to the critical rad
/*                                      to determine if the sat lies
/*                                      above or below the platform
/*                                      artificial horizon.
/*      OrbitInView                      Is the satellite ever above the
/*                                      horizon plain of the platform?
/*                                      (IE, is the orbit itself, regard
/*                                      less of the satellite present
/*                                      position, it view? YES=1, NO=0.
/*
/*  COMPILER:       Borland C++ Builder3 Standard version
/*                  This compiler should be used to compile and link.
*****/

```

```

/*
/*****
void CompareOrbit( struct Satellite &Sat,
                  struct Aircraft &Platform,
                  double ThetaGInRad,
                  int    &OrbitInView,
                  double &CriticalRadius,
                  double &SatRadius,
                  ErrorStructure &ErrorList);

#endif

```

## B.8 The ABLPA Preprocessor

The ABL Predictive Avoidance Preprocessor is the culmination of the modules discussed in this chapter thus far. The purpose of the Predictive Avoidance Preprocessor is to read the Two-Line Element (TLE) input file and screen it to pick out any satellites which could not be within range of the ABL platform for a set time in the future. The TLE set is an input file that contains a list of all satellites for which the user has a concern. Each satellite is either in the ABL engagement area, or outside that area. The preprocessor returns a shortened TLE input file that contains only those satellites that are within the engagement area. Unfortunately, the Main Processor must execute very quickly, in a real-time operational role. Therefore, the number of satellites that it needs

Aircraft Information		Greenwich Time		Orionwich Maiden Reference	
Aircraft Latitude Degrees	10 N	Calculation Year	1998	Reference Hour	4
Aircraft Latitude Minutes	0	Calculation Month (1-12)	8	Reference Minute	40
Aircraft Latitude Seconds	0	Calculation Day	14	Reference Second	1.299
Aircraft Longitude Degrees	55	Calculation Hour	3	Modified Julian Date	11029.0
Aircraft Longitude Minutes	0	Calculation Minute	58	Seconds To Next Run	60
Aircraft Longitude Seconds	0	Calculation Second	2.0		
Aircraft Altitude	12.9				
Aircraft Xdot (ECEF) km/h	0	Input/Output Files Input TLE File to Preprocessor: PAData/TLEFile4.txt Output TLE File to Processor: PAData/PreprocessorOutput.txt			
Aircraft Ydot (ECEF) km/h	300				
Aircraft Zdot (ECEF) km/h	0				

Satellites In Range (SSC #)	Error Message Box
No satellites in range	No Errors

Number Of Satellites Evaluated	0	Evaluate TLE File
Number of Satellites In Range	0	

**Figure B.7. The Graphical Interface to the Preprocessor**



to process should be as small as possible. The preprocessor ensures that this is so. This “screening”, in turn, reduces the execution time of the Main Processor. The execution time of the preprocessor is not an issue, because the preprocessor can be run at any time, and there is no need to run the preprocessor in a given time slot. Despite this fact, the ABLPA preprocessor generally runs in under one second. This time estimate is for running with a standard desktop 200 MHz computer. The Graphical Interface developed for the Preprocessor is shown in Figure B.7. Of course, as with all of the modules described in this chapter, the user of these libraries could easily create their own graphical (or non-graphical) interface, designed to their own tastes. This interface is simply provided to make use of the preprocessor more convenient. Notice that most of the input and output is handled via TLE files. The format of a standard TLE file is given in *Appendix F*. The final output file resulting from the run of the preprocessor will serve as input file for the Main Processor. The next chapter will describe the nature of the Main Processor and the way in which this output file will be put to use.

### B.8.1 Inputs

**char InFileName[MAXNAMELENGTH]** This parameter holds the name of the Two Line Element Set that holds the satellites to be evaluated.

**char OutFileName[MAXNAMELENGTH]** Holds the name of the file to which the output satellites’ Two-Line Element set information is routed to. This file holds all of the satellites that have been judged by the Preprocessor to be “in view” of the platform.

**struct Aircraft &ABLPlatform** ABLPlatform is a structure of type “Aircraft” that holds all of the information about the position of the aircraft at the time of execution of the Preprocessor.

**int ReferenceHour** Reference hour Refers to the Reference angle of  $\theta_g$  (The angle between the Greenwich meridian and the Vernal Equinox). This

angles is given in hours, minutes and seconds as opposed to degrees or radians. This parameter holds the hours portion of  $\theta_g$ .

**int ReferenceMinute** The minutes portion of  $\theta_g$ .

**double ReferenceSecond** The seconds portion of  $\theta_g$ .

**double RefModJulianDate** This parameter holds the Modified Julian Date at which the reference angle,  $\theta_g$ , was taken. This allows  $\theta_g$  to be propagated forward to the present moment.

**int CalcYear** The current year.

**int CalcMonth** The current month (1-12).

**int CalcDay** The current day (1-31).

**int CalcHour** The current hour (1-24).

**int CalcMinute** The current minute (1-60).

**double CalcSecond** The current second. This is the only part of the current time that can be given as a non-integer. This field should be accurate to at least three decimal places.

**double TimeToNextRun** The estimated time until the next run of the Preprocessor.

**ErrorStructure &ErrorList** This parameter is both an input and output parameter. Each module uses it to assess whether a fatal error has occurred somewhere else in the program, and uses it to record errors that may be important to the user.

### B.8.2 Outputs

**int &InFileLength** This parameter tells the user how many elements were read in from the file specified by the input parameter "InFileName[MAXNAMELENGTH]". This is the total number of satellites that were evaluated during the run of the Preprocessor.

**int &OutFileLength** This parameter tells the user how many elements were written to the file specified by the input parameter "OutFileName[MAXNAMELENGTH]". This is the total number of satellites that were judged to be "in-view" of the platform between the time of the run and the next run of the preprocessor.

**double   &ThetaGInDegrees** This is the instantaneous angle between the Greenwich meridian and the Vernal Eqinox at the moment of execution of the preprocessor.

**ErrorStructure       &ErrorList** This parameter is both an input and output parameter. Each module uses it to assess whether a fatal error has occurred somewhere else in the program, and uses it to record errors which may be important to the user.

### B.8.3 The PAPreprocessor.h Header File

```

/*****
/*  MODULE NAME:      PAPreprocessor
/*  AUTHOR:          Captain David Vloedman
/*  DATE CREATED:    August 18, 1998
/*
/*
/*  PURPOSE:         This set of modules supports/composes the preprocessor
/*                   used to evaluate whether or not the satellites are ever
/*                   above the platform horizon.
/*
/*
/*  COMPILER:        Borland C++ Builder3 Standard version
/*                   This compiler should be used to compile and link.
/*
/*
/*****
#endif PAPreprocessorH
#define PAPreprocessorH

/*****
/*****          FUCTIONS          *****/
/*****

/*****
/*  FUNCTION NAME:    PAPreprocessor
/*  AUTHOR:          Captain David Vloedman
/*  DATE CREATED:    October 6, 1998
/*
/*
/*  PURPOSE:         This procedure will read in an input file of Two Line
/*                   Element (TLE) sets and perform an analysis to determine
/*                   whether or not they are within view of the airborne
/*                   platform. If a satellite is in view, it will be added
/*                   to the ouput file, which is the input file for the main
/*                   processor.
/*
/*
/*  INPUTS:          NAME:          DEFINITION:
/*                   InFileName     Holds name of the satellite file*
/*                   OutFileName     File that holds the sats in view*
/*                   InFileLength    The total number
/*                   ABLPlatform     Holds all information about ABL
/*                   ReferenceHour   Platform position/disposition
/*                   ReferenceMinute This holds the value of Theta G
/*                   ReferenceSecond at RefModJulianDate. The angle
/*                   RefModJulianDate of Theta G is given in hours,
/*                                   minutes, and seconds instead of
/*                                   degrees, where 24 hrs = 360 deg
/*                   CalcYear        Holds the minutes of Theta G at
/*                   Calcmonth       RefModJulianDate.
/*                   CalcDay         Holds the seconds of Theta G at
/*                   CalcHour        RefModJulianDate.
/*                   CalcMinute     This is the reference date when
/*                   CalcSecond     an actual observation of the
/*                                   true value of theta G was made.
/*                                   Holds the current calender year
/*                                   Holds the Calender month(1 - 12)
/*                                   Holds calender day
/*                                   Holds the calender hour
/*                                   Holds the calender minute
/*                                   Holds the calender second

```

```

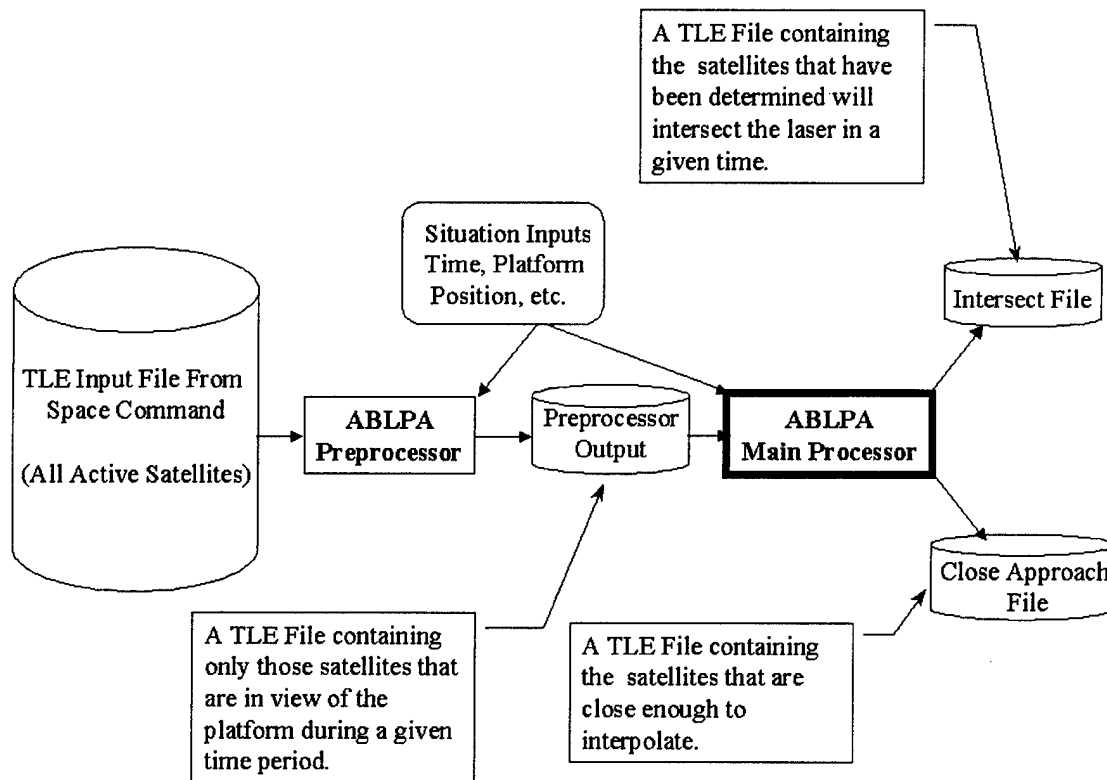
/*          TimeToNextRun          The amount of time for which the*/
/*          current run must last. This is */
/*          To determine how much time in */
/*          seconds will transpire before */
/*          next update is received.      */
/*                                     */
/* OUTPUTS:      NAME:      DESCRIPTION:      */
/*          InFileLength    The total number of satellites */
/*          that have been evaluated in the */
/*          InFile          */
/*          OutFileLength    The total number of satellites */
/*          that are in view of the platform*/
/*          and have been put in the outfile*/
/*          ThetaGInDegrees  The rotation angle between the */
/*          Earth's current ECEF position */
/*          and its ECI position.      */
/*          ErrorList        Errors that have occurred      */
/*          THE FINAL OUTPUT IS THE ACTUAL OUTFILE ITSELF WHICH IS */
/*          WRITTEN DIRECTLY TO DISK SO IT CAN BE ACCESSED BY THE */
/*          MAIN PROCESSOR.      */
/*          COMPILER:      Borland C++ Builder3 Standard version */
/*          This compiler should be used to compile and link.      */
/*          */
/*****
PAPreprocessor( char    InFileName[MAXNAMELENGTH],
                char    OutFileName[MAXNAMELENGTH],
                int      &InFileLength,
                int      &OutFileLength,
                struct    Aircraft &Platform,
                int       ReferenceHour,
                int       ReferenceMinute,
                double    ReferenceSecond,
                double    RefModJulianDate,
                int       CalcYear,
                int       CalcMonth,
                int       CalcDay,
                int       CalcHour,
                int       CalcMinute,
                double    CalcSecond,
                double    TimeToNextRun,
                double    &ThetaGInDegrees,
                ErrorStructure &ErrorList);

```

#endif

## Appendix C. ABLPA Main Processor Software Implementation

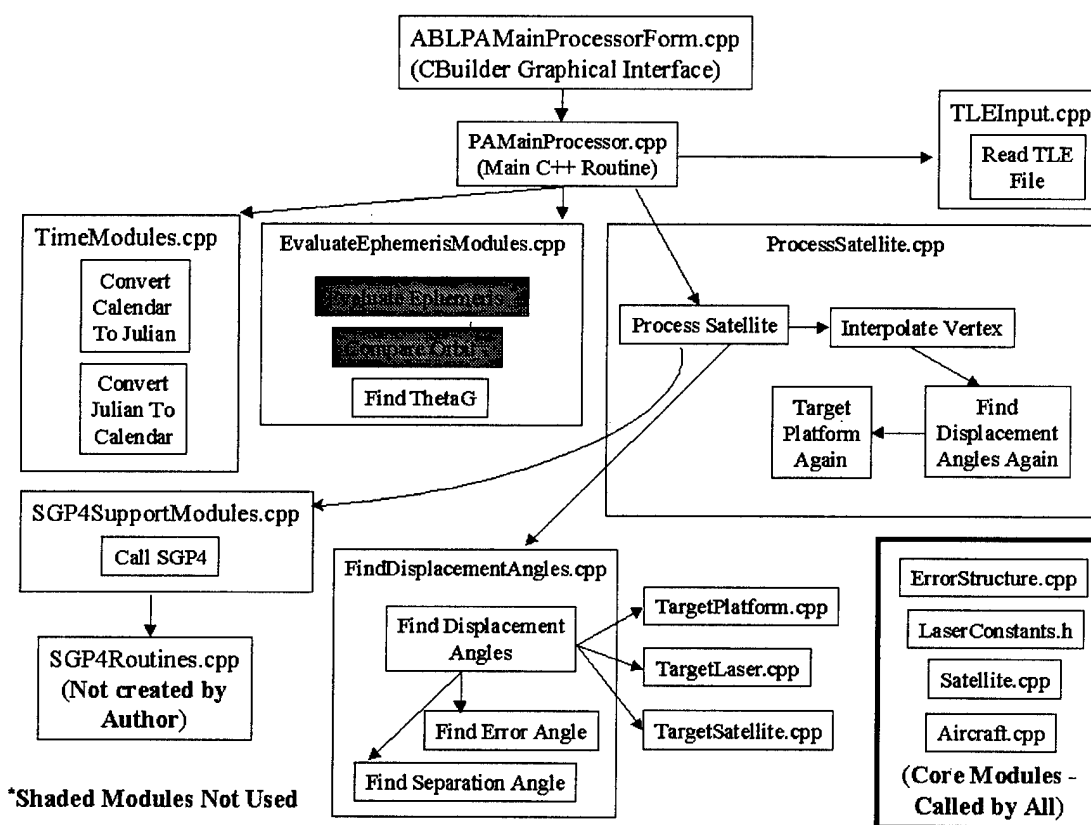
The Airborne Laser Predictive Avoidance (ABL-PA) Main Processor is the second of two software packages developed in this project. Figure C.1 illustrates how the Main Processor fits into the overall hierarchy of the software.



**Figure C.1 Where the ABLPA Main Processor Fits in the Software Hierarchy**

It can be seen that the task of the Main Processor is to take the output file containing all active satellites in view of the platform, and create two output files. The first output file will contain all of the satellites that are forecast to be intersected by the laser during the laze duration. The Processor also creates an output file containing the satellites that pass closely enough to the laser path to be “interpolated”, or analyzed, but may or may not

actually intersect the laser. This second “close-approach” file is used more for testing and verification than operational use. Statistically, more often than not the Intersection File will be empty after a full Main Processor run-through, because the chances of “hitting” a satellite (even with a theoretical error-angle) is slim. This close approach file can be used to verify the successful run and processing of the Main Processor. Both output files have exactly the same format as the main TLE file, except they have fewer (or no) satellites within them.



**Figure C.2 ABLPA Main Processor Calling Tree**

### C.1 Main Processor Modular Format

The Processor is a conglomeration of many software libraries that were created and tested independently before being combined to form the Processor. Figure C.2

shows the basic modules that comprise the processor, and their grouping into “libraries”. Each module and library shown will be explained within this chapter. Five of the libraries shown in the Figure have already been discussed in Chapter V. These libraries

**Table C.1 The Six Remaining Libraries Composing the ABLPA Main Processor**

<b>Sub-Project Title</b>	<b>Modules Tested (C++)</b>	<b>GUI Interface Module (C++ Builder 3)</b>	<b>Purpose</b>
<b>ABLPA Main Processor</b>	All processor modules as shown in Figure C.2	PAMainProcessorForm	To provide a user-friendly way to run the processor
<b>Target Platform</b>	TargetPlatform	TargetPlatformForm	To find the instantaneous position, velocity and acceleration of the platform, and generate the REN conversion matrix
<b>Target Laser</b>	TargetLaser	TestTargetLaserForm	To find the position, velocity and acceleration vectors of the laser
<b>Target Satellite</b>	TargetSatellite	TargetSatelliteForm	To get position and velocity of satellite from SGP4, and compute satellites acceleration
<b>Find Displacement Angles</b>	FindDisplacementAngles FindErrorAngle FindSeparationAngle	FindDisplacementAngle-Form	To find the separation angle between the laser and the satellite
<b>Process Satellite</b>	ProcessSatellite FindDisplacementAngles-Again TargetPlatformAgain	ProcessSatelliteForm	To pull together all of the other modules and completely process one satellite, first forecasting a close approach angle, then interpolating, if necessary



are the Core Modules, the Error Structure, TLE Input, Time Modules, SGP4 Support Modules, and the Evaluate Ephemeris Modules. They will not be covered again here. The libraries that have not yet been discussed are listed in Table C.1. Each of these libraries are also a project in and of themselves, tested using the GUI as seen in the table. The discussion of the preprocessor will progress through each of these libraries individually, discussing the nature of the function served by the software library, as well as comments on each module within that library. The interfaces and input/output parameters used with each module will be emphasized. The actual code for each module in the ABLPA Main Processor will be listed out in *Appendix D*. The code for each library-testing GUI interface will be listed in *Appendix E*. Only the "Header File" or the files with the ".h" extension will be listed here in the discussion, because they are short and contain important interface information that should be discussed. All of the implementation code will be included in their respective Appendices.

## **C.2 The Target Platform Library**

The Target platform Library holds the module responsible for processing all information pertaining to the platform. This module is TargetPlatform. The TargetPlatform module can be run by itself using the Graphical User Interface (GUI) created as a front-end to the module for testing purposes. This GUI, Shown in Figure C.3, is run using the Borland C++ Builder module TargetPlatformForm.

## C.2.1 TargetPlatform

The TargetPlatform Module serves three main functions. The first task performed by this module is to find the instantaneous position, velocity, and acceleration of the platform at the time specified, in the ECI coordinate frame. The second task is to calculate the elements of the ECI-to-REN conversion matrix. Recall that the Radial-East-North (REN) coordinate frame rotates with the platform, and so should be found when

The screenshot shows a graphical interface for testing the TargetPlatform module. It is organized into several panels:

- Aircraft Information:** Contains input fields for Aircraft Latitude Degrees (10), Aircraft Latitude Minutes (0), Aircraft Longitude Degrees (55), Aircraft Longitude Minutes (0), Aircraft Altitude (12.9), Aircraft X-ECF (km) (-300), Aircraft Y-ECF (km) (50), and Aircraft Z-ECF (km) (100). A dropdown menu for 'N' is visible.
- Geosynchronous Orbit Parameters:** Contains input fields for Reference Hour (4), Reference Minute (40), Reference Second (1.299), and Modified Julian Date (11029.0).
- Platform Velocity ECI:** Contains input fields for Position Vector ECI (X, Y, Z) and Velocity Vector ECI (Vx, Vy, Vz), all set to 0.
- Platform Velocity REN:** Contains input fields for Position Vector REN (X, Y, Z) and Velocity Vector REN (Vx, Vy, Vz), all set to 0.
- Aircraft Time Parameters:** Contains input fields for Year (1998), Month (8), Day (14), Hour (3), Minute (58), and Second (2.0).
- No Errors:** A large text box at the bottom left indicating no errors.

**Figure C.3. The Graphical Interface Used to Test TargetPlatform**

other platform calculations are made. This conversion matrix, which is passed element by element in the parameter list (for clarity), is used to rotate all of the satellite vectors to the REN frame as well, and so must be made available to the TargetSatellite library as

well. The third function performed by TargetPlatform is to rotate each of its output ECI platform motion vectors into the REN frame, and output the resulting REN motion vectors in the parameter list as well. This makes for a rather long and unsightly parameter list. However, this is preferable to incorporating everything into a larger structure, to allow a more “instructive” interface.

### Inputs

**struct Aircraft &ABLPlatform** ABLPlatform is a structure of type “Aircraft” that holds all of the information about the position of the aircraft at the time of execution of the Preprocessor.

**double ThetaGInRad** This is the angle at which the Earth’s Greenwich Meridian is currently at with respect to the ECI frame, where the referent angle is the Vernal Equinox. This angle should be in radians.

**double JulianDate** This is the modified Julian Date (The Julian Date – 2440000) that needs to be propagated to. This is the actual time at which the user wishes to find the position of the satellite.

### Outputs

**double &PlatformECIRhoX** The current ECI X position vector of the position of the platform with respect to the center of the Earth, as derived from the latitude and longitude given in the Aircraft structure input.

**double &PlatformECIRhoY** The current ECI Y position vector of the position of the platform with respect to the center of the Earth, as derived from the latitude and longitude given in the Aircraft structure input.

**double &PlatformECIRhoZ** The current ECI Z position vector of the position of the platform with respect to the center of the Earth, as derived from the latitude and longitude given in the Aircraft structure input.

**double &PlatformECIRhoXDot** The current ECI X velocity vector of the velocity of the platform with respect to the center of the Earth, as derived from the ECEF velocities given in the Aircraft structure input.

**double &PlatformECIRhoYDot** The current ECI Y velocity vector of the velocity of the platform with respect to the center of the Earth, as derived from the ECEF velocities given in the Aircraft structure input.

**double &PlatformECIRhoZDot** The current ECI Z velocity vector of the velocity of the platform with respect to the center of the Earth, as derived from the ECEF velocities given in the Aircraft structure input.

**double &PlatformECIRhoXDotDot** The current ECI X acceleration vector of the acceleration of the platform with respect to the center of the Earth, as derived from the centripetal and Coriolis accelerations only. It is assumed that the aircraft itself is maintaining straight and level flight at constant velocity.

**double &PlatformECIRhoYDotDot** The current ECI Y acceleration vector of the acceleration of the platform with respect to the center of the Earth, as derived from the centripetal and Coriolis accelerations only. It is assumed that the aircraft itself is maintaining straight and level flight at constant velocity.

**double &PlatformECIRhoZDotDot** The current ECI Z acceleration vector of the acceleration of the platform with respect to the center of the Earth, as derived from the centripetal and Coriolis accelerations only. It is assumed that the aircraft itself is maintaining straight and level flight at constant velocity.

**double &PlatformRENrhoR** The current REN R (Radial) position vector of the position of the platform with respect to the center of the Earth, as derived from the ECI position vector, rotated into the REN frame.

**double &PlatformRENrhoE** The current REN E (East) position vector of the position of the platform with respect to the center of the Earth, as derived from the ECI position vector, rotated into the REN frame.

**double &PlatformRENrhoN** The current REN N (North) position vector of the position of the platform with respect to the center of the Earth, as derived from the ECI position vector, rotated into the REN frame.

**double &PlatformRENrhoRDot** The current REN R (Radial) velocity vector of the velocity of the platform with respect to the center of the Earth, as derived from the ECI velocity vector, rotated into the REN frame.

**double &PlatformRENrhoEDot** The current REN E (East) velocity vector of the velocity of the platform with respect to the center of the Earth, as derived from the ECI velocity vector, rotated into the REN frame.

**double &PlatformRENrhoNDot** The current REN N (North) velocity vector of the velocity of the platform with respect to the center of the Earth, as derived from the ECI velocity vector, rotated into the REN frame.

**double &PlatformRENRRhoRDotDot** The current REN R acceleration vector of the acceleration of the platform with respect to the center of the Earth, as derived from the centripetal and Coriolis accelerations derived in the ECI frame and rotated into the REN frame. It is assumed that the aircraft itself is maintaining straight and level flight at constant velocity.

**double &PlatformRENRRhoEDotDot** The current REN E acceleration vector of the acceleration of the platform with respect to the center of the Earth, as derived from the centripetal and Coriolis accelerations derived in the ECI frame and rotated into the REN frame. It is assumed that the aircraft itself is maintaining straight and level flight at constant velocity.

**double &PlatformRENRRhoNDotDot** The current REN N acceleration vector of the acceleration of the platform with respect to the center of the Earth, as derived from the centripetal and Coriolis accelerations derived in the ECI frame and rotated into the REN frame. It is assumed that the aircraft itself is maintaining straight and level flight at constant velocity.

**double &ECItORENMatrix11** This an element (row 1, column 1) of the matrix used to transform a vector from the ECI coordinate frame to the REN coordinate frame.

**double &ECItORENMatrix12** This an element (row 1, column 2) of the matrix used to transform a vector from the ECI coordinate frame to the REN coordinate frame.

**double &ECItORENMatrix13** This an element (row 1, column 3) of the matrix used to transform a vector from the ECI coordinate frame to the REN coordinate frame.

**double &ECItORENMatrix21** This an element (row 2, column 1) of the matrix used to transform a vector from the ECI coordinate frame to the REN coordinate frame.

**double &ECItORENMatrix22** This an element (row 2, column 2) of the matrix used to transform a vector from the ECI coordinate frame to the REN coordinate frame.

**double &ECItORENMatrix23** This an element (row 2, column 3) of the matrix used to transform a vector from the ECI coordinate frame to the REN coordinate frame.

**double &ECItORENMatrix31** This an element (row 3, column 1) of the matrix used to transform a vector from the ECI coordinate frame to the REN coordinate frame.

**double &ECItToRENMatrix32** This an element (row 3, column 2) of the matrix used to transform a vector from the ECI coordinate frame to the REN coordinate frame.

**double &ECItToRENMatrix33** This an element (row 3, column 3) of the matrix used to transform a vector from the ECI coordinate frame to the REN coordinate frame.

**ErrorStructure &ErrorList** This is the error-handling structure that is used as both an input (to see if errors have occurred) and an output (to list any errors occurring in this module).

## C.2.2 The TargetPlatform.h Header File

```
#ifndef TargetPlatformH
#define TargetPlatformH
/*****
/*  MODULE NAME:      TargetPlatform.cpp
/*  AUTHOR:          Captain David Vloedman
/*  DATE CREATED:    January 13, 1998
/*
/*  PURPOSE:         This set of modules supports the processor and are
/*                   used to establish the platform's position, velocity, and
/*                   acceleration wrt the platform in the REN frame. The
/*                   ECI to REN conversion matrix is also passed back to
/*                   allow other conversions later.
/*
/*  COMPILER:        Borland C++ Builder3 Standard version
/*                   This compiler should be used to compile and link.
/*
*****/
/*****
/* C++BUILDER-SPECIFIC LIBRARIES */
*****/
#include <vcl.h>
#pragma hdrstop
#pragma package(smart_init)
/*****
/* USER-BUILT LIBRARIES
*****/
#include "LaserConstants.h"
#include "Aircraft.h"
#include "ErrorStructure.h"
#include "TargetPlatform.h"
/*****
/* C STANDARD LIBRARIES
*****/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <iostream.h>
#include <conio.h>
#include <math.h>
```

```

/*****
/*****          FUCTIONS          *****/
/*****

/*****
/*  FUNCTION NAME:  TargetPlatform      */
/*  AUTHOR:        Captain David Vloedman  */
/*  DATE CREATED:   November 17, 1998      */
/*                                          */
/*  PURPOSE:       This function will take the position of the aircraft and */
/*                  position, velocity and acceleration in the REN frame of */
/*                  the Airborn laser platform.                               */
/*                                          */
/*  INPUTS:        NAME:                DEFINITION:                          */
/*                                          */
/*                  ABLPlatform          Holds all information about ABL */
/*                                          Platform position/disposition */
/*                  JulianDate           The time to which the position */
/*                                          of sat should be propagated to */
/*  OUTPUTS:       NAME:                DESCRIPTION:                         */
/*                                          */
/*                  PlatformECIRhoX      X magnitude in ECI frame at Jul */
/*                                          date of X pos vector          */
/*                  PlatformECIRhoY      Y magnitude in ECI frame at Jul */
/*                                          date of Y pos vector          */
/*                  PlatformECIRhoZ      Z magnitude in ECI frame at Jul */
/*                                          date of Z pos vector          */
/*                  PlatformECIRhoXDot   X magnitude in ECI frame at Jul */
/*                                          date of X vel vector          */
/*                  PlatformECIRhoYDot   Y magnitude in ECI frame at Jul */
/*                                          date of Y vel vector          */
/*                  PlatformECIRhoZDot   Z magnitude in ECI frame at Jul */
/*                                          date of Z vel vector          */
/*                  PlatformECIRhoXDotDot X magnitude in ECI frame at Jul */
/*                                          date of X acc vector          */
/*                  PlatformECIRhoYDotDot Y magnitude in ECI frame at Jul */
/*                                          date of Y acc vector          */
/*                  PlatformECIRhoZDotDot Z magnitude in ECI frame at Jul */
/*                                          date of Z acc vector          */
/*                  PlatformRENrhoR      Radial component in Radial, East */
/*                                          North coordinate frame of the */
/*                                          Rho vector descibed above in the */
/*                                          ECI frame                      */
/*                  PlatformRENrhoE      East component in Radial, East */
/*                                          North coordinate frame of the */
/*                                          Rho vector descibed above in the */
/*                                          ECI frame                      */
/*                  PlatformRENrhoN      North component in Radial, East */
/*                                          North coordinate frame of the */
/*                                          Rho vector descibed above in the */
/*                                          ECI frame                      */
/*                  PlatformRENrhoRDot   Radial Velocity in Radial, East */
/*                                          North coordinate frame of the */
/*                                          Rho vector descibed above in the */
/*                                          ECI frame                      */
/*                  PlatformRENrhoEDot   East velocity in Radial, East */
/*                                          North coordinate frame of the */
/*                                          Rho vector descibed above in the */
/*                                          ECI frame                      */
/*                  PlatformRENrhoNDot   North velocity in Radial, East */
/*                                          North coordinate frame of the */
/*                                          Rho vector descibed above in the */
/*                                          ECI frame                      */
/*                  PlatformRENrhoRDotDot Radial accel. in Radial, East */

```

```

/*          North coordinate frame of the */
/*          Rho vector descibed above in the*/
/*          ECI frame */
/*          PlatformRENRhoEDotDot      East accel. in Radial, East */
/*          North coordinate frame of the */
/*          Rho vector descibed above in the*/
/*          ECI frame */
/*          PlatformRENRhoNDotDot      North accel. in Radial, East */
/*          North coordinate frame of the */
/*          Rho vector descibed above in the*/
/*          ECI frame */
/*          ECItORENMatrixXY           The ECI to REN conversion matrix*/
/*          ErrorList                  The Errors which have occurred */
/*          */
/*  COMPILER:      Borland C++ Builder3 Standard version */
/*          This compiler should be used to compile and link. */
/*          */
/*****
void TargetPlatform(struct Aircraft &Platform,
                    double &ThetaGInRad,
                    double JulianDate,
                    double &PlatformECIRhoX,
                    double &PlatformECIRhoY,
                    double &PlatformECIRhoZ,
                    double &PlatformECIRhoXDot,
                    double &PlatformECIRhoYDot,
                    double &PlatformECIRhoZDot,
                    double &PlatformECIRhoXDotDot,
                    double &PlatformECIRhoYDotDot,
                    double &PlatformECIRhoZDotDot,
                    double &PlatformRENRhoR,
                    double &PlatformRENRhoE,
                    double &PlatformRENRhoN,
                    double &PlatformRENRhoRDot,
                    double &PlatformRENRhoEDot,
                    double &PlatformRENRhoNDot,
                    double &PlatformRENRhoRDotDot,
                    double &PlatformRENRhoEDotDot,
                    double &PlatformRENRhoNDotDot,
                    double &ECItORENMatrix11,
                    double &ECItORENMatrix12,
                    double &ECItORENMatrix13,
                    double &ECItORENMatrix21,
                    double &ECItORENMatrix22,
                    double &ECItORENMatrix23,
                    double &ECItORENMatrix31,
                    double &ECItORENMatrix32,
                    double &ECItORENMatrix33,
                    ErrorStructure &ErrorList);
#endif

```



### C.3 The Target Laser Library

The Target Laser Library houses all code that transforms the lasers inputs, in terms of azimuth and acceleration, into a position, velocity and acceleration vector in the REN coordinate frame. TargetLaser, the module in this library responsible for accomplishing this task, can be run independently from the Processor using the Graphical Interface shown in Figure C.4. This interface is handled by the C++ Builder module, TargetLaserForm, the code for which is listed in *Appendix E*.

INPUTS		OUTPUTS	
Laser Az and Elevation in deg		Laser Position in REN Frame	
Laser Azimuth	75.0	R Dot (m)	0
Laser Elevation	20.0	E Dot (m)	0
Change in Az and Elevation in deg/sec		N Dot (m)	0
Rate Az Dot	1.0	Laser Velocity in REN Frame	
Rate Elev Dot	1.0	R Dot Dot (m/sec)	0
Accel of Az and Elevation in deg/sec^2		E Dot Dot (m/sec)	0
Azimuth Dot Dot	0.1	N Dot Dot (m/sec)	0
Elevation Dot Dot	0.1	Laser Acceleration in REN Frame	
Target Laser		R Dot Dot Dot (m/sec^2)	0
		E Dot Dot Dot (m/sec^2)	0
		N Dot Dot Dot (m/sec^2)	0
No Errors			

**Figure C.4.** GUI Used to Run and Test TargetLaser

#### C.3.1 TargetLaser

TargetLaser is the module responsible for finding the lasers turrets position, velocity, and acceleration in the REN frame. As such, it takes inputs in terms of azimuth

and acceleration, and converts to a unit position vector, a velocity vector, and an acceleration vector in the REN frame.

### Inputs

**double AzimuthInDegrees** The current azimuth of the laser turret in degrees.

**double ElevationInDegrees** The current elevation of the laser turret in degrees.

**double AzimuthDot** The current rate of change of the azimuth of the laser turret in degrees per second.

**double ElevationDot** The current rate of change of the elevation of the laser turret in degrees per second.

**double AzimuthDotDot** The current acceleration of the azimuth of the laser turret in degrees per second.

**double ElevationDotDot** The current acceleration of the elevation of the laser turret in degrees per second.

### Outputs

**double &LaserRENrhoRPtr** The current REN R (Radial) unit position vector of the position of the laser with respect to the platform, as derived from the azimuth and elevation.

**double &LaserRENrhoEPtr** The current REN E (East) unit position vector of the position of the laser with respect to the platform, as derived from the azimuth and elevation.

**double &LaserRENrhoNPtr** The current REN N (North) unit position vector of the position of the laser with respect to the platform, as derived from the azimuth and elevation

**double &LaserRENrhoRDotPtr** The current REN R (Radial) velocity vector of the velocity of the laser with respect to the platform, as derived from the azimuth, elevation and the rate of change of the azimuth and elevation.

**double &LaserRENrhoEDotPtr** The current REN E (East) velocity vector of the velocity of the laser with respect to the platform, as derived from the azimuth, elevation and the rate of change of the azimuth and elevation.

**double &LaserRENrhoNDotPtr** The current REN N (North) velocity vector of the velocity of the laser with respect to the platform, as derived from the azimuth, elevation and the rate of change of the azimuth and elevation.

**double &LaserRENrhoRDotDotPtr** The current REN R (Radial) acceleration vector of the acceleration of the laser with respect to the platform, as derived from the azimuth, elevation, the rate of change of the azimuth and elevation, and the acceleration of the azimuth and elevation.

**double &LaserRENrhoEDotDotPtr** The current REN E (East) acceleration vector of the acceleration of the laser with respect to the platform, as derived from the azimuth, elevation, the rate of change of the azimuth and elevation, and the acceleration of the azimuth and elevation.

**double &LaserRENrhoNDotDotPtr** The current REN N (North) acceleration vector of the acceleration of the laser with respect to the platform, as derived from the azimuth, elevation, the rate of change of the azimuth and elevation, and the acceleration of the azimuth and elevation.

**ErrorStructure &ErrorList** This is the error-handling structure that is used as both an input (to see if errors have occurred) and an output (to list any errors occurring in this module).

### C.3.2 The TargetLaser.h Header File

```
#ifndef TargetLaserH
#define TargetLaserH
/*****
/*  MODULE NAME:      TargetLaser.cpp                      */
/*  AUTHOR:          Captain David Vloedman                */
/*  DATE CREATED:    January 11, 1999                      */
/*                                                          */
/*  PURPOSE:         This set of modules supports the processor and are
/*                   used to evaluate whether or not the satellite is ever
/*                   above the platform horizon.            */
/*                                                          */
/*  COMPILER:        Borland C++ Builder3 Standard version */
/*                   This compiler should be used to compile and link.
/*                                                          */
/*****
/*****
/* C++BUILDER-SPECIFIC LIBRARIES */
/*****
#include <vcl.h>
```

```

#pragma hdrstop
#pragma package(smart_init)
/*****
/* USER-BUILT LIBRARIES */
*****/
#include "LaserConstants.h"
#include "ErrorStructure.h"
#include "TargetLaser.h"
/*****
/* C STANDARD LIBRARIES */
*****/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <iostream.h>
#include <conio.h>
#include <math.h>

/*****
***** FUCTIONS *****
*****/

/*****
/* FUNCTION NAME: TargetLaser */
/* AUTHOR: Captain David Vloedman */
/* DATE CREATED: January 3, 1999 */
/* */
/* PURPOSE: This routine finds the unit direction vector of the */
/* laser turret given its reported azimuth and elevation. */
/* */
/* INPUTS: NAME: DEFINITION: */
/* Azimuth This is the Azimuth (reported in */
/* degrees east of north) of the */
/* laser turret. */
/* Elevation This is the Elevation (reported */
/* in degrees above horizon) of the */
/* laser turret. */
/* AzimuthDot This is the Azimuth rate of */
/* change of the laser turret. */
/* AzimuthDot This is the Elevation rate of */
/* change of the laser turret. */
/* AzimuthDotDot This is the Azimuth acceleration */
/* of the laser turret. */
/* AzimuthDotDot This is the Elevation accel. */
/* of the laser turret. */
/* */
/* OUTPUTS: NAME: DESCRIPTION: */
/* LaserRENrhoR The unit Radial component of the */
/* position vector given in the */
/* REN (Radial, East, North) coord */
/* frame which is centered on the */
/* platform. */
/* LaserRENrhoE The unit East component of the */
/* position vector given in the */
/* REN (Radial, East, North) coord */
/* frame which is centered on the */
/* platform. */
/* LaserRENrhoN The unit North component of the */
/* position vector given in the */
/* REN (Radial, East, North) coord */
/* frame which is centered on the */
/* platform. */
/* LaserRENrhoDot The unit Radial velocity of the */

```

```

/*          position vector given in the */
/*          REN (Radial, East, North) coord */
/*          frame which is centered on the */
/*          platform. */
/*          LaserRENrhoEDot      The unit East velocity of the */
/*          position vector given in the */
/*          REN (Radial, East, North) coord */
/*          frame which is centered on the */
/*          platform. */
/*          LaserRENrhoNDot      The unit North velocity of the */
/*          position vector given in the */
/*          REN (Radial, East, North) coord */
/*          frame which is centered on the */
/*          platform. */
/*          LaserRENrhoDotDot     The unit Radial accel. of the */
/*          position vector given in the */
/*          REN (Radial, East, North) coord */
/*          frame which is centered on the */
/*          platform. */
/*          LaserRENrhoEDotDot    The unit East accel. of the */
/*          position vector given in the */
/*          REN (Radial, East, North) coord */
/*          frame which is centered on the */
/*          platform. */
/*          LaserRENrhoNDotDot    The unit North accel. of the */
/*          position vector given in the */
/*          REN (Radial, East, North) coord */
/*          frame which is centered on the */
/*          platform. */
/*          ErrorList            The Errors which have occurred */
/*          */
/*  COMPILER:      Borland C++ Builder3 Standard version */
/*                  This compiler should be used to compile and link. */
/*          */
/*****
void TargetLaser(double AzimuthInDegrees,
                 double ElevationInDegrees,
                 double AzimuthDot,
                 double ElevationDot,
                 double AzimuthDotDot,
                 double ElevationDotDot,
                 double &LaserRENrhoRPtr,
                 double &LaserRENrhoEPtr,
                 double &LaserRENrhoNPtr,
                 double &LaserRENrhoRDotPtr,
                 double &LaserRENrhoEDotPtr,
                 double &LaserRENrhoNDotPtr,
                 double &LaserRENrhoRDotDotPtr,
                 double &LaserRENrhoEDotDotPtr,
                 double &LaserRENrhoNDotDotPtr,
                 ErrorStructure &ErrorList);

#endif

```

## C.4 The Target Satellite Library

The Target Satellite Library houses the code that locates the satellite and its parameters of motion with the help of an interface to SGP4. The module responsible for accomplishing this task is TargetSatellite. this module can be run independently, if desired, using the GUI shown in Figure C.5. This GUI is run using the C++ Builder module, TargetSatelliteForm, the listing for which can be seen in *Appendix E*.

The GUI is organized into several sections:

- Aircraft Information:**
  - Aircraft Latitude Degrees: 10 N
  - Aircraft Latitude Minutes: 0
  - Aircraft Latitude Seconds: 0
  - Aircraft Longitude Degrees: 55
  - Aircraft Longitude Minutes: 0
  - Aircraft Longitude Seconds: 0
  - Aircraft Altitude (km): 12.9
  - Aircraft AOA (ECEF) km/h: 0
  - Aircraft Ydot (ECEF) km/h: 300
  - Aircraft Zdot (ECEF) km/h: 0
- Greenwich Mean Time:**
  - Reference Hour: 4
  - Reference Minute: 40
  - Reference Second: 1.299
  - Modified Julian Date: 11029.0
- Greenwich Time Reference:**
  - Calculation Year: 1998
  - Calculation Month (1-12): 8
  - Calculation Day: 14
  - Calculation Hour: 3
  - Calculation Minute: 58
  - Calculation Second: 2.0
- Target Information:**
  - Target Name: 0
  - Target Color: 0
  - Target ID: 0
  - Epoch Year: 0
  - Epoch Day: 0
  - Target Day: 0
  - Target Day Offset: 0
  - Epoch Day: 0
  - Epoch Type: 0
  - Epoch Name: 0
  - Epoch: 0
  - Right Ascension: 0
  - Declination: 0
  - Argument of Perigee: 0
  - Mean Anomaly: 0
  - Mean Motion: 0
  - True ALE Epoch: 0
- Buttons:**
  - Populate Using File
  - PAData/TLEFile5.txt
  - Find ECI Parameters
- ECI TESTING CRITERIA:**
  - Satellite Position ECI:**
    - ECI X (km): 0
    - ECI Y (km): 0
    - ECI Z (km): 0
  - Satellite Velocity ECI:**
    - X Dot (km/sec): 0
    - Y Dot (km/sec): 0
    - Z Dot (km/sec): 0
  - Satellite Acceleration ECI:**
    - X Dot Dot (m/s^2): 0
    - Y Dot Dot (m/s^2): 0
    - Z Dot Dot (m/s^2): 0
- REN TESTING CRITERIA:**
  - Satellite Position REN:**
    - REN R (km): 0
    - REN E (km): 0
    - REN H (km): 0
  - Satellite Velocity REN:**
    - R Dot (km/sec): 0
    - E Dot (km/sec): 0
    - H Dot (km/sec): 0
  - Satellite Acceleration REN:**
    - R Dot Dot (m/s^2): 0
    - E Dot Dot (m/s^2): 0
    - H Dot Dot (m/s^2): 0
- Status:** No Errors

Figure C.5. Graphical Interface for TargetSatellite

### C.4.1 TargetSatellite

The Target Satellite module must handle three functions. First, it must interface with SGP4 (via the SGP4 Support modules) and obtain a satellite's position and velocity as of the Julian Date given, in the ECI frame. Second, based on the position of the satellite in the ECI frame, the acceleration of the satellite in the ECI frame must also be calculated. Third, TargetSatellite must take the ECI-to-REN conversion matrix formulated by TargetPlatform, and convert the satellite position, velocity and accelerations in the ECI frame to the appropriate vectors in the REN frame.

#### Inputs

**struct Satellite &Sat** The structure holding all of the satellite ephemeris information. The Type "Satellite" is defined in Satellite.h.

**double ThetaGInRad** This is the angle at which the Earth's Greenwich Meridian is currently at with respect to the ECI frame, where the referent angle is the Vernal Equinox. This angle should be in radians.

**double &ECitoRENMatrix11** This an element (row 1, column 1) of the matrix used to transform a vector from the ECI coordinate frame to the REN coordinate frame.

**double &ECitoRENMatrix12** This an element (row 1, column 2) of the matrix used to transform a vector from the ECI coordinate frame to the REN coordinate frame.

**double &ECitoRENMatrix13** This an element (row 1, column 3) of the matrix used to transform a vector from the ECI coordinate frame to the REN coordinate frame.

**double &ECitoRENMatrix21** This an element (row 2, column 1) of the matrix used to transform a vector from the ECI coordinate frame to the REN coordinate frame.

**double &ECitoRENMatrix22** This an element (row 2, column 2) of the matrix used to transform a vector from the ECI coordinate frame to the REN coordinate frame.

**double &ECitoRENMatrix23** This an element (row 2, column 3) of the matrix used to transform a vector from the ECI coordinate frame to the REN coordinate frame.

**double &ECitoRENMatrix31** This an element (row 3, column 1) of the matrix used to transform a vector from the ECI coordinate frame to the REN coordinate frame.

**double &ECItORENMatrix32** This an element (row 3, column 2) of the matrix used to transform a vector from the ECI coordinate frame to the REN coordinate frame.

**double &ECItORENMatrix33** This an element (row 3, column 3) of the matrix used to transform a vector from the ECI coordinate frame to the REN coordinate frame.

## Outputs

**double &SatECIRhoX** The current ECI X position vector of the position of the satellite with respect to the center of the Earth, as derived from SGP4.

**double &SatECIRhoY** The current ECI Y position vector of the position of the satellite with respect to the center of the Earth, as derived from SGP4.

**double &SatECIRhoZ** The current ECI Z position vector of the position of the satellite with respect to the center of the Earth, as derived from SGP4.

**double &SatECIRhoXDot** The current ECI X velocity vector of the velocity of the satellite with respect to the center of the Earth, as derived from SGP4.

**double &SatECIRhoYDot** The current ECI Y velocity vector of the velocity of the satellite with respect to the center of the Earth, as derived from SGP4.

**double &SatECIRhoZDot** The current ECI Z velocity vector of the velocity of the satellite with respect to the center of the Earth, as derived from SGP4.

**double &SatECIRhoXDotDot** The current ECI X acceleration vector of the acceleration of the satellite with respect to the center of the Earth, as derived from the position of the satellite found by SGP4.

**double &SatECIRhoYDotDot** The current ECI Y acceleration vector of the acceleration of the satellite with respect to the center of the Earth, as derived from the position of the satellite found by SGP4.

**double &SatECIRhoZDotDot** The current ECI Z acceleration vector of the acceleration of the satellite with respect to the center of the Earth, as derived from the position of the satellite found by SGP4.



**double &SatRENrhoR** The current REN R (Radial) position vector of the position of the satellite with respect to the center of the Earth, as derived from the ECI position vector, found by SGP4, rotated into the REN frame.

**double &SatRENrhoE** The current REN E (East) position vector of the position of the satellite with respect to the center of the Earth, as derived from the ECI position vector, found by SGP4, rotated into the REN frame.

**double &SatRENrhoN** The current REN N (North) position vector of the position of the satellite with respect to the center of the Earth, as derived from the ECI position vector, found by SGP4, rotated into the REN frame.

**double &SatRENrhoRDot** The current REN R (Radial) velocity vector of the velocity of the satellite with respect to the center of the Earth, as derived from the ECI velocity vector, found by SGP4, rotated into the REN frame.

**double &SatRENrhoEDot** The current REN E (East) velocity vector of the velocity of the satellite with respect to the center of the Earth, as derived from the ECI velocity vector, found by SGP4, rotated into the REN frame.

**double &SatRENrhoNDot** The current REN N (North) velocity vector of the velocity of the satellite with respect to the center of the Earth, as derived from the ECI velocity vector, found by SGP4, rotated into the REN frame.

**double &SatRENrhoRDotDot** The current REN R acceleration vector of the acceleration of the satellite with respect to the center of the Earth, as derived from the acceleration in the ECI frame.

**double &SatRENrhoEDotDot** The current REN E acceleration vector of the acceleration of the satellite with respect to the center of the Earth, as derived from the acceleration in the ECI frame.

**double &SatRENrhoNDotDot** The current REN N acceleration vector of the acceleration of the satellite with respect to the center of the Earth, as derived from the acceleration in the ECI frame.

**ErrorStructure      &ErrorList** This is the error-handling structure that is used as both an input (to see if errors have occurred) and an output (to list any errors occurring in this module).

## C.4.2 The TargetSatellite.h Header File

```

#ifndef TargetSatelliteH
#define TargetSatelliteH
/*****
/*  MODULE NAME:      TargetSatellite.cpp                      */
/*  AUTHOR:          Captain David Vloedman                    */
/*  DATE CREATED:    November 17, 1998                         */
/*                                                           */
/*  PURPOSE:         This set of modules supports the preprocessor and are */
/*                   used to establish the satellites position, velocity, and */
/*                   acceleration wrt the platform in the REN frame.          */
/*                                                           */
/*  COMPILER:        Borland C++ Builder3 Standard version    */
/*                   This compiler should be used to compile and link.       */
/*                                                           */
*****/
/*****
/* C++BUILDER-SPECIFIC LIBRARIES */
*****/
#include <vcl.h>
#pragma hdrstop
#pragma package(smart_init)
/*****
/* USER-BUILT LIBRARIES          */
*****/
#include "TimeModules.h"
#include "TLEInput.h"
#include "LaserConstants.h"
#include "Satellite.h"
#include "Aircraft.h"
#include "ErrorStructure.h"
#include "EvaluateEphemerisModules.h"
#include "SGP4SupportModules.h"
#include "TargetSatellite.h"
/*****
/* C STANDARD LIBRARIES          */
*****/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <iostream.h>
#include <conio.h>
#include <math.h>

/*****
*****          FUCTIONS          *****
*****/

/*****
/*  FUNCTION NAME:    TargetSatellite                          */
/*  AUTHOR:          Captain David Vloedman                    */
/*  DATE CREATED:    November 17, 1998                         */
/*                                                           */
/*  PURPOSE:         This function will take the position of the aircraft and */
/*                   the orbital elements of the satellite and calculate      */
/*                   the azimuth and elevation of the satellite from the      */
/*                   Airborn laser platform.                               */
/*                                                           */
/*  INPUTS:          NAME:          DEFINITION:                */
/*                   Sat            Holds all ephemeris information */
/*                   for the Satellite being studied              */

```

```

/*          JulianDate          The time to which the position */
/*          of sat should be propagated to */
/*          ECItoreNMatrix(RowCol) The ECI to REN conversion matrix*/
/*          THIS IS USED TO CONVERT FROM ECI*/
/*          COORDINATE FRAME TO THE RADIAL, */
/*          EAST, NORTH (REN) FRAME. */
/* OUTPUTS:  NAME:  DESCRIPTION: */
/*          SatECIRhoX      X magnitude in ECI frame at Jul */
/*          date of sat radial vector - the */
/*          platform radial position vector */
/*          SatECIRhoY      Y magnitude in ECI frame at Jul */
/*          date of sat radial vector - the */
/*          platform radial position vector */
/*          SatECIRhoZ      Z magnitude in ECI frame at Jul */
/*          date of sat radial vector - the */
/*          platform radial position vector */
/*          SatECIRhoXDot    X velocity in ECI frame at Jul */
/*          date of sat radial vector - vel */
/*          in X axis direction. */
/*          SatECIRhoYDot    Y velocity in ECI frame at Jul */
/*          date of sat radial vector - vel */
/*          in Y axis direction. */
/*          SatECIRhoZDot    Z velocity in ECI frame at Jul */
/*          date of sat radial vector - vel */
/*          in Z axis direction. */
/*          SatECIRhoXDotDot  X accel. in ECI frame at Jul */
/*          date of sat radial vector - acc.*/
/*          in X axis direction. */
/*          SatECIRhoYDotDot  Y accel. in ECI frame at Jul */
/*          date of sat radial vector - acc.*/
/*          in Y axis direction. */
/*          SatECIRhoZDotDot  Z accel. in ECI frame at Jul */
/*          date of sat radial vector - acc.*/
/*          in Z axis direction. */
/*          SatRENrhoR      The Radial Component of the */
/*          position vector of the satellite*/
/*          wrt Earth center in the REN */
/*          coordinate frame. */
/*          SatRENrhoE      The East Component of the */
/*          position vector of the satellite*/
/*          wrt Earth center in the REN */
/*          coordinate frame. */
/*          SatRENrhoN      The North Component of the */
/*          position vector of the satellite*/
/*          wrt Earth center in the REN */
/*          coordinate frame. */
/*          SatRENrhoRDot    The Radial Component of the */
/*          velocity vector of the satellite*/
/*          wrt Earth center in the REN */
/*          coordinate frame. */
/*          SatRENrhoEDot    The East Component of the */
/*          velocity vector of the satellite*/
/*          wrt Earth center in the REN */
/*          coordinate frame. */
/*          SatRENrhoNDot    The North Component of the */
/*          velocity vector of the satellite*/
/*          wrt Earth center in the REN */
/*          coordinate frame. */
/*          SatRENrhoRDotDot  The Radial Component of the */
/*          accel vector of the satellite */
/*          wrt Earth center in the REN */
/*          coordinate frame. */
/*          SatRENrhoEDotDot  The East Component of the

```

```

/*          accel vector of the satellite */
/*          wrt Earth center in the REN */
/*          coordinate frame. */
/*          SatRENrhoNDotDot          The North Component of the */
/*          accel vector of the satellite */
/*          wrt Earth center in the REN */
/*          coordinate frame. */
/*          ErrorList          The Errors which have occurred */
/*          */
/*  COMPILER:          Borland C++ Builder3 Standard version */
/*          This compiler should be used to compile and link. */
/*          */
/*****
void TargetSatellite(struct Satellite &Sat,
    double JulianDate,
    double ECitoRENMatrix11,
    double ECitoRENMatrix12,
    double ECitoRENMatrix13,
    double ECitoRENMatrix21,
    double ECitoRENMatrix22,
    double ECitoRENMatrix23,
    double ECitoRENMatrix31,
    double ECitoRENMatrix32,
    double ECitoRENMatrix33,
    double &SatECIRhoX,
    double &SatECIRhoY,
    double &SatECIRhoZ,
    double &SatECIRhoXDot,
    double &SatECIRhoYDot,
    double &SatECIRhoZDot,
    double &SatECIRhoXDotDot,
    double &SatECIRhoYDotDot,
    double &SatECIRhoZDotDot,
    double &SatRENrhoR,
    double &SatRENrhoE,
    double &SatRENrhoN,
    double &SatRENrhoRDot,
    double &SatRENrhoEDot,
    double &SatRENrhoNDot,
    double &SatRENrhoRDotDot,
    double &SatRENrhoEDotDot,
    double &SatRENrhoNDotDot,
    ErrorStructure &ErrorList);

#endif

```

## C.5 The Find Displacement Angle Modules Library

The modules of the Find Displacement Angle Modules Library are responsible for finding the angle that separates the laser turret unit direction position vector and the satellite position vector. This includes finding the separation angle of these two vectors (as seen from the platform), finding the rate of change of this separation angle, and finding the acceleration of this angle. This library also finds the error angle contributed by position errors in the forecast. The three modules contained in this library are FindDisplacementAngles, FindErrorAngle, and FindSeparationAngle. These three modules can be tested using the GUI seen in Figure C.6.

[illegible]

**Figure C.6. GUI Used to Run and Test FindDisplacementAngles Module**

### C.5.1 FindDisplacementAngles

FindDisplacementAngles is the primary Module in this library. It calls both FindErrorAngle and FindSeparationAngle. It also calls each of the libraries already discussed. This module first locates the laser turret, the platform, and the satellite in REN space (using the libraries above). It then subtracts the platform position, velocity and acceleration from the satellite's position, velocity and acceleration to locate the platform at the origin of the REN coordinate frame with respect to the satellite. This sets up the laser turret motion vectors and the satellite motion vectors in the REN frame with respect to the platform, and allows the separation angle to be found more easily. A call to FindSeparationAngle finds the separation angle, the rate of change of the angle and the acceleration of the angle between the laser unit position vector and the satellite position vector. A call is also made to FindErrorAngle to find the half-error angle resulting from position uncertainties in the satellite, platform, and missile.

#### Inputs

**struct Satellite &Sat** The structure holding all of the satellite ephemeris information. The Type "Satellite" is defined in Satellite.h.

**struct Aircraft &ABLPlatform** ABLPlatform is a structure of type "Aircraft" that holds all of the information about the position of the aircraft at the time of execution of the Preprocessor.

**double ThetaGInRad** This is the angle at which the Earth's Greenwich Meridian is currently at with respect to the ECI frame, where the referent angle is the Vernal Equinox. This angle should be in radians.

**double JulianDate** This is the modified Julian Date (The Julian Date - 2440000) that needs to be propagated to. This is the actual time at which the user wishes to find the position of the satellite.

**double AzimuthInDegrees** The current azimuth of the laser turret in degrees.

**double ElevationInDegrees** The current elevation of the laser turret in degrees.

**double AzimuthDot** The current rate of change of the azimuth of the laser turret in degrees per second.

**double ElevationDot** The current rate of change of the elevation of the laser turret in degrees per second.

**double AzimuthDotDot** The current acceleration of the azimuth of the laser turret in degrees per second.

**double ElevationDotDot** The current acceleration of the elevation of the laser turret in degrees per second.

**double SatPositionErrorInMeters** The radius of the “sphere” inside which the satellite is known to reside (in meters) throughout the forecast period (around 10 to 30 seconds). For SGP4, this may be as high as around 10 km (10000 m).

**double PlatformPositionErrorInMeters** The radius of the “sphere” inside which the platform is known to reside (in meters) throughout the forecast period (around 10 to 30 seconds). For a 747 on autopilot with GPS tracking, a rough estimate might be 50 meters.

**double MissilePositionErrorInMeters** The radius of the “sphere” inside which the missile is known to reside (in meters) throughout the forecast period (around 10 to 30 seconds). This may just be an educated guess. It is difficult to now the behavior of the missile based on initial conditions, and this parameter will have to be given some thought.

**double RangeToMissileInKilometers** The range from the platform to the missile.

**double OtherErrorAngleInDeg** This is a “catch-all” parameter, to be used if, in the future, there are other error angles that crop up that have not already been accounted for.

## Outputs

**double &PlatformSatRENrhoR** The current REN R (Radial) position vector of the position of the satellite with respect to the platform, as derived from the ECI position vector, found by SGP4, rotated into the REN frame, and subtracted by the platform REN position vector.

**double &PlatformSatRENrhoE** The current REN E (East) position vector of the position of the satellite with respect to the platform, as derived from the ECI position vector, found by SGP4, rotated into the REN frame, and subtracted by the platform REN position vector.

**double &PlatformSatRENrhoN** The current REN N (North) position vector of the position of the satellite with respect to the platform, as derived from the ECI position vector, found by SGP4, rotated into the REN frame, and subtracted by the platform REN position vector.

**double &PlatformSatRENrhoRDot** The current REN R (Radial) velocity vector of the velocity of the satellite with respect to the platform, as derived from the ECI velocity vector, found by SGP4, rotated into the REN frame and subtracted by the platform REN velocity vector

**double &PlatformSatRENrhoEDot** The current REN E (East) velocity vector of the velocity of the satellite with respect to the platform, as derived from the ECI velocity vector, found by SGP4, rotated into the REN frame and subtracted by the platform REN velocity vector

**double &PlatformSatRENrhoNDot** The current REN N (North) velocity vector of the velocity of the satellite with respect to the platform, as derived from the ECI velocity vector, found by SGP4, rotated into the REN frame and subtracted by the platform REN velocity vector

**double &PlatformSatRENrhoRDotDot** The current REN R acceleration vector of the acceleration of the satellite with respect to the platform, as derived from the acceleration in the ECI frame, rotated into the REN frame, subtracted by the platform REN acceleration

**double &PlatformSatRENrhoEDotDot** The current REN E acceleration vector of the acceleration of the satellite with respect to the platform, as derived from the acceleration in the ECI frame, rotated into the REN frame, subtracted by the platform REN acceleration

**double &PlatformSatRENrhoNDotDot** The current REN N acceleration vector of the acceleration of the satellite with respect to the platform, as derived from the acceleration in the ECI frame, rotated into the REN frame, subtracted by the platform REN acceleration

**double &LaserRENrhoR** The current REN R (Radial) unit position vector of the position of the laser with respect to the platform, as derived from the azimuth and elevation.



**double &LaserRENrhoE** The current REN E (East) unit position vector of the position of the laser with respect to the platform, as derived from the azimuth and elevation.

**double &LaserRENrhoN** The current REN N (North) unit position vector of the position of the laser with respect to the platform, as derived from the azimuth and elevation

**double &LaserRENrhoRDot** The current REN R (Radial) velocity vector of the velocity of the laser with respect to the platform, as derived from the azimuth, elevation and the rate of change of the azimuth and elevation.

**double &LaserRENrhoEDot** The current REN E (East) velocity vector of the velocity of the laser with respect to the platform, as derived from the azimuth, elevation and the rate of change of the azimuth and elevation.

**double &LaserRENrhoNDot** The current REN N (North) velocity vector of the velocity of the laser with respect to the platform, as derived from the azimuth, elevation and the rate of change of the azimuth and elevation.

**double &LaserRENrhoRDotDot** The current REN R (Radial) acceleration vector of the acceleration of the laser with respect to the platform, as derived from the azimuth, elevation, the rate of change of the azimuth and elevation, and the acceleration of the azimuth and elevation.

**double &LaserRENrhoEDotDot** The current REN E (East) acceleration vector of the acceleration of the laser with respect to the platform, as derived from the azimuth, elevation, the rate of change of the azimuth and elevation, and the acceleration of the azimuth and elevation.

**double &LaserRENrhoNDotDot** The current REN N (North) acceleration vector of the acceleration of the laser with respect to the platform, as derived from the azimuth, elevation, the rate of change of the azimuth and elevation, and the acceleration of the azimuth and elevation.

**double &RangeToSatInKilometers** Range to the satellite from the platform in kilometers.

**double &ErrorAngleInRadians** The final error angle (in radians) resulting from the position errors given previously.

**double &SeparationAngle** The separation angle between the laser turret unit position vector and the satellite position vector in the REN frame.

**double &SepAngleDot** The rate of change of the separation angle in radians per second.

**double &SepAngleDotDot** The acceleration of the separation angle in radians per second squared.

**ErrorStructure      &ErrorList** This is the error-handling structure that is used as both an input (to see if errors have occurred) and an output (to list any errors occurring in this module).

### C.5.2 FindErrorAngle

The module, FindErrorAngle is the module responsible for taking the position uncertainty errors for the platform, missile, and satellite, along with the ranges to the missile and satellite, and finding the error angle imposed by each uncertainty. It then takes the sum of the squares of each of these error angles to find the resulting overall error angle.

#### Inputs

**double &RangeToSatInKilometers** Range to the satellite from the platform in kilometers.

**double SatPositionErrorInMeters** The radius of the “sphere” inside which the satellite is known to reside (in meters) throughout the forecast period (around 10 to 30 seconds). For SGP4, this may be as high as around 10 km (10000 m).

**double PlatformPositionErrorInMeters** The radius of the “sphere” inside which the platform is known to reside (in meters) throughout the forecast period (around 10 to 30 seconds). For a 747 on autopilot with GPS tracking, a rough estimate might be 50 meters.

**double MissilePositionErrorInMeters** The radius of the “sphere” inside which the missile is known to reside (in meters) throughout the forecast period (around 10 to 30 seconds). This may just be an educated guess. It is difficult to now the behavior of the missile based on initial conditions, and this parameter will have to be given some thought.

**double RangeToMissileInKilometers** The range from the platform to the missile.

**double OtherErrorAngleInDeg** This is a “catch-all” parameter, to be used if, in the future, there are other error angles that crop up that have not already been accounted for.

### Outputs

**double &ErrorAngleInRadians** The final error angle (in radians) resulting from the position errors given previously.

**ErrorStructure &ErrorList** This is the error-handling structure that is used as both an input (to see if errors have occurred) and an output (to list any errors occurring in this module).

### **C.5.3 FindSeparationAngle**

FindSeparationAngle is the module responsible for actually calculating the angle between the unit position vector for the laser turret and the position vector of the satellite (from the platform) in the REN frame. It is also responsible for finding the rate of change and the acceleration of the change of this separation angle.

### Inputs

**double LaserRENrhoR** The current REN R (Radial) unit position vector of the position of the laser with respect to the platform, as derived from the azimuth and elevation.

**double LaserRENrhoE** The current REN E (East) unit position vector of the position of the laser with respect to the platform, as derived from the azimuth and elevation.

**double LaserRENrhoN** The current REN N (North) unit position vector of the position of the laser with respect to the platform, as derived from the azimuth and elevation

**double LaserRENrhoRDot** The current REN R (Radial) velocity vector of the velocity of the laser with respect to the platform, as derived from the azimuth, elevation and the rate of change of the azimuth and elevation.

**double LaserRENrhoEDot** The current REN E (East) velocity vector of the velocity of the laser with respect to the platform, as derived from the azimuth, elevation and the rate of change of the azimuth and elevation.

**double LaserRENrhoNDot** The current REN N (North) velocity vector of the velocity of the laser with respect to the platform, as derived from the azimuth, elevation and the rate of change of the azimuth and elevation.

**double LaserRENrhoRDotDot** The current REN R (Radial) acceleration vector of the acceleration of the laser with respect to the platform, as derived from the azimuth, elevation, the rate of change of the azimuth and elevation, and the acceleration of the azimuth and elevation.

**double LaserRENrhoEDotDot** The current REN E (East) acceleration vector of the acceleration of the laser with respect to the platform, as derived from the azimuth, elevation, the rate of change of the azimuth and elevation, and the acceleration of the azimuth and elevation.

**double LaserRENrhoNDotDot** The current REN N (North) acceleration vector of the acceleration of the laser with respect to the platform, as derived from the azimuth, elevation, the rate of change of the azimuth and elevation, and the acceleration of the azimuth and elevation.

**double SatRENrhoR** The current REN R (Radial) position vector of the position of the satellite with respect to the platform, as derived from the ECI position vector, found by SGP4, rotated into the REN frame, and subtracted by the platform REN position vector.

**double SatRENrhoE** The current REN E (East) position vector of the position of the satellite with respect to the platform, as derived from the ECI position vector, found by SGP4, rotated into the REN frame, and subtracted by the platform REN position vector.

**double SatRENrhoN** The current REN N (North) position vector of the position of the satellite with respect to the platform, as derived from the ECI position vector, found by SGP4, rotated into the REN frame, and subtracted by the platform REN position vector.

**double SatRENrhoRDot** The current REN R (Radial) velocity vector of the velocity of the satellite with respect to the platform, as derived from the ECI velocity vector, found by SGP4, rotated into the REN frame and subtracted by the platform REN velocity vector

**double SatRENrhoEDot** The current REN E (East) velocity vector of the velocity of the satellite with respect to the platform, as derived from the ECI velocity vector, found by SGP4, rotated into the REN frame and subtracted by the platform REN velocity vector

**double SatRENrhoNDot** The current REN N (North) velocity vector of the velocity of the satellite with respect to the platform, as derived from the ECI

velocity vector, found by SGP4, rotated into the REN frame and subtracted by the platform REN velocity vector

**double SatRENrhoRDotDot** The current REN R acceleration vector of the acceleration of the satellite with respect to the platform, as derived from the acceleration in the ECI frame, rotated into the REN frame, subtracted by the platform REN acceleration

**double SatRENrhoEDotDot** The current REN E acceleration vector of the acceleration of the satellite with respect to the platform, as derived from the acceleration in the ECI frame, rotated into the REN frame, subtracted by the platform REN acceleration

**double SatRENrhoNDotDot** The current REN N acceleration vector of the acceleration of the satellite with respect to the platform, as derived from the acceleration in the ECI frame, rotated into the REN frame, subtracted by the platform REN acceleration

### Outputs

**double &SeparationAngle** The separation angle between the laser turret unit position vector and the satellite position vector in the REN frame.

**double &SepAngleDot** The rate of change of the separation angle in radians per second.

**double &SepAngleDotDot** The acceleration of the separation angle in radians per second squared.

**ErrorStructure      &ErrorList** This is the error-handling structure that is used as both an input (to see if errors have occurred) and an output (to list any errors occurring in this module).

## C.5.4 The FindDisplacementAngleModules.h Header File

```

#ifndef FindDisplacementAngleModulesH
#define FindDisplacementAngleModulesH
/*****
/* MODULE NAME:      FindDisplacementAngleModules.h          */
/* AUTHOR:           Captain David Vloedman                  */
/* DATE CREATED:     3 January, 1999                          */
/*                                                           */
/* PURPOSE:          This set of modules supports the Main Processor and are */
/*                   used to evaluate the error angle and the displacement */
/*                   angle between the laser position vector in the REN frame*/
/*                   and the satellite position vector in the same frame.   */
/*                                                           */
/* COMPILER:         Borland C++ Builder3 Standard version   */
/*                   This compiler should be used to compile and link.      */
/*                                                           */
*****/
/*****
/* C++BUILDER-SPECIFIC LIBRARIES */
*****/
#include <vcl.h>
#pragma hdrstop
#pragma package(smart_init)
/*****
/* USER-BUILT LIBRARIES          */
*****/
#include "TimeModules.h"
#include "TLEInput.h"
#include "LaserConstants.h"
#include "Satellite.h"
#include "Aircraft.h"
#include "ErrorStructure.h"
#include "EvaluateEphemerisModules.h"
#include "SGP4SupportModules.h"
#include "TargetSatellite.h"
#include "TargetPlatform.h"
#include "TargetLaser.h"
/*****
/* C STANDARD LIBRARIES          */
*****/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <iostream.h>
#include <conio.h>
#include <math.h>

/*****
/* ***** FUCTIONS ***** */
*****/

/*****
/* FUNCTION NAME:  FindDisplacementAngles                      */
/* AUTHOR:         Captain David Vloedman                    */
/* DATE CREATED:   January 3, 1999                           */
/*                                                           */
/* PURPOSE:        This function will take satellite and platform data and */
/*                 will use it to find the error angle and the displacement */
/*                 angle between the laser position vector in the REN frame*/
/*                 and the satellite position vector in the same frame.     */
*****/

```

```

/*
/* INPUTS:      NAME:      DEFINITION:
/*
/*      Sat      Holds all ephemeris information
/*               for the Satellite being studied
/*
/*      ABLPlatform  Holds all information about ABL
/*               Platform position/disposition
/*
/*      JulianDate  The time to which the position
/*               of sat should be propagated to
/*
/*      ThetaGInRadians  The angle between the Greenwich
/*               Meridian and the Vernal Equinox
/*               at JulianDate.
/*
/*      LazerAzimuthInDegrees  Lazer Azimuth at Laze Start time
/*               in Degrees
/*
/*      LazerAzimuthDot  The rate of change of the Az
/*               in Degrees/Sec.
/*
/*      LazerAzimuthDotDot  The rate of change of the rate
/*               of change of the Azimuth (Accel)
/*               in Degrees/Sec^2
/*
/*      LazerElevationInDegrees  Lazer Elevation at Laze Start
/*               in Degrees
/*
/*      LazerElevationDot  The rate of change of the El
/*               in Degrees/Sec.
/*
/*      LazerElevationDotDot  The rate of change of the rate
/*               of change of the Elevat. (Accel)
/*               in Degrees/Sec^2
/*
/*      SatPositionErrorInMeters  Holds the radius of the error
/*               spheroid that describes the
/*               area in which the satellite is
/*               known to exist (in meters).
/*
/*      PlatformPositionError...  Holds the radius of the error
/*               spheroid that describes the
/*               area in which the platform is
/*               known to exist (in meters).
/*
/*      MissilePositionError...  Holds the radius of the error
/*               spheroid that describes the
/*               area in which the missile is
/*               known to exist (in meters).
/*
/*      RangeToMissileInKilo...  The Range to the missile (km)
/*
/*      OtherErrorAnglesInDeg  Holds any other error angles
/*               (in degrees) that may be a
/*               significant source of error.
/*               This should usually be set to
/*               zero (0.0) float.
/*
/* OUTPUTS:      NAME:      DESCRIPTION:
/*
/*      PlatformSatRENrhoR  The Radial Component of the
/*               position vector of the satellite
/*               wrt the platform in the REN
/*               coordinate frame.
/*
/*      PlatformSatRENrhoE  The East Component of the
/*               position vector of the satellite
/*               wrt the platform in the REN
/*               coordinate frame.
/*
/*      PlatformSatRENrhoN  The North Component of the
/*               position vector of the satellite
/*               wrt the platform in the REN
/*               coordinate frame.
/*
/*      PlatformSatRENrhoRDot  The Radial Component of the
/*               velocity vector of the satellite
/*               wrt the platform in the REN
/*               coordinate frame.
/*
/*      PlatformSatRENrhoEDot  The East Component of the
/*               velocity vector of the satellite
/*               wrt the platform in the REN

```

```

/*                                coordinate frame.                                */
/*                                PlatformSatRENrhoNDot The North Component of the */
/*                                velocity vector of the satellite */
/*                                wrt the platform in the REN */
/*                                coordinate frame.                                */
/*                                PlatformSatRENrhoRDotDot The Radial Component of the */
/*                                accel vector of the satellite */
/*                                wrt the platform in the REN */
/*                                coordinate frame.                                */
/*                                PlatformSatRENrhoEDotDot The East Component of the */
/*                                accel vector of the satellite */
/*                                wrt the platform in the REN */
/*                                coordinate frame.                                */
/*                                PlatformSatRENrhoNDotDot The North Component of the */
/*                                accel vector of the satellite */
/*                                wrt the platform in the REN */
/*                                coordinate frame.                                */
/*                                LaserRENrhoR The Radial unit direction of the */
/*                                lazer beam trajectory in the REN */
/*                                frame.                                            */
/*                                LaserRENrhoE The East unit direction of the */
/*                                lazer beam trajectory in the REN */
/*                                frame.                                            */
/*                                LaserRENrhoN The North unit direction of the */
/*                                lazer beam trajectory in the REN */
/*                                frame.                                            */
/*                                LaserRENrhoRDot The Radial unit velocity of the */
/*                                lazer beam trajectory in the REN */
/*                                frame in unit dirXradians/sec */
/*                                LaserRENrhoEDot The East unit velocity of the */
/*                                lazer beam trajectory in the REN */
/*                                frame in unit dirXradians/sec */
/*                                LaserRENrhoNDot The North unit velocity of the */
/*                                lazer beam trajectory in the REN */
/*                                frame in unit dirXradians/sec */
/*                                LaserRENrhoRDotDot The Radial unit accel. of the */
/*                                lazer beam trajectory in the REN */
/*                                frame in unit dirXradians/sec^2 */
/*                                LaserRENrhoEDotDot The East unit accel. of the */
/*                                lazer beam trajectory in the REN */
/*                                frame in unit dirXradians/sec^2 */
/*                                LaserRENrhoNDotDot The North unit accel. of the */
/*                                lazer beam trajectory in the REN */
/*                                frame in unit dirXradians/sec^2 */
/*                                RangeInKilometers Holds the range of the aircraft */
/*                                to the satellite in kilometers. */
/*                                ErrorAngleInRadians The total error angle in radians */
/*                                SeparationAngle The separation (in radians) of */
/*                                the LaserRENrho and */
/*                                PlatformSatRENrho vectors. */
/*                                SeparationAngleDot The rate of change (in rad/sec) */
/*                                of the separation of LaserRENrho */
/*                                PlatformSatRENrho vectors. */
/*                                SeparationAngleDotDot The acceleration (in rad/sec^2) */
/*                                of the separation of LaserRENrho */
/*                                and PlatformSatRENrho vectors. */
/*                                ErrorList The Errors which have occurred */
/*                                */
/*                                COMPILER: Borland C++ Builder3 Standard version */
/*                                This compiler should be used to compile and link. */
/*                                */
/*****
void FindDisplacementAngles(struct Aircraft &Platform,

```



```

struct Satellite &Sat,
double &ThetaGInRad,
double JulianDate,
double LaserAzimuthInDegrees,
double LaserAzimuthDot,
double LaserAzimuthDotDot,
double LaserElevationInDegrees,
double LaserElevationDot,
double LaserElevationDotDot,
double SatPositionErrorInMeters,
double PlatformPositionErrorInMeters,
double MissilePositionErrorInMeters,
double RangeToMissileInKilometers,
double OtherErrorAngleInDeg,
double &PlatformSatRENrhoR,
double &PlatformSatRENrhoE,
double &PlatformSatRENrhoN,
double &PlatformSatRENrhoRDot,
double &PlatformSatRENrhoEDot,
double &PlatformSatRENrhoNDot,
double &PlatformSatRENrhoRDotDot,
double &PlatformSatRENrhoEDotDot,
double &PlatformSatRENrhoNDotDot,
double &LaserRENrhoR,
double &LaserRENrhoE,
double &LaserRENrhoN,
double &LaserRENrhoRDot,
double &LaserRENrhoEDot,
double &LaserRENrhoNDot,
double &LaserRENrhoRDotDot,
double &LaserRENrhoEDotDot,
double &LaserRENrhoNDotDot,
double &RangeToSatInKilometers,
double &ErrorAngleInRadians,
double &SeparationAngle,
double &SepAngleDot,
double &SepAngleDotDot,
ErrorStructure &ErrorList);

```

```

/*****
/* FUNCTION NAME: FindErrorAngle */
/* AUTHOR: Captain David Vloedman */
/* DATE CREATED: January 3, 1999 */
/* */
/* PURPOSE: This function will take the range to satellite and the */
/* satellite position error and find the appropriate error */
/* error angle. */
/* */
/* INPUTS: NAME: DEFINITION: */
/* Range Holds the range of the aircraft */
/* to the satellite in kilometers. */
/* SatPositionErrorInMeters Holds the radius of the error */
/* spheroid that describes the */
/* area in which the satellite is */
/* known to exist (in meters). */
/* PlatformPositionError... Holds the radius of the error */
/* spheroid that describes the */
/* area in which the platform is */
/* known to exist (in meters). */
/* MissilePositionError... Holds the radius of the error */
/* spheroid that describes the */
/* area in which the missile is */
*****/

```

```

/*          known to exist (in meters).          */
/*          RangeToMissileInKilo... The Range to the missile (km) */
/*          OtherErrorAnglesInDeg Holds any other error angles    */
/*          (in degrees) that may be a              */
/*          significant source of error.            */
/*          This should usually be set to          */
/*          zero (0.0) float.                      */
/* OUTPUTS:      NAME:      DESCRIPTION:          */
/*          ErrorAngleInRadians The total error angle in radians */
/*          ErrorList          The Errors which have occurred    */
/*          */
/* COMPILER:      Borland C++ Builder3 Standard version          */
/*          This compiler should be used to compile and link.    */
/*          */
/*****
void FindErrorAngle(double RangeToSatInKilometers,
                    double SatPositionErrorInMeters,
                    double PlatformPositionErrorInMeters,
                    double MissilePositionErrorInMeters,
                    double RangeToMissileInKilometers,
                    double OtherErrorAnglesInDeg,
                    double &ErrorAngleInRadians,
                    ErrorStructure &ErrorList);
*****/

/*****
/* FUNCTION NAME: FindSeparationAngle          */
/* AUTHOR:      Captain David Vloedman        */
/* DATE CREATED: January 3, 1999              */
/*          */
/* PURPOSE:     This routine finds the angle separating the satellite */
/*          position vector and the laser turret unit direction      */
/*          vector in the REN coordinate frame, as well as the rate  */
/*          of change and the acceleration of that separation.        */
/*          */
/* INPUTS:      NAME:      DEFINITION:          */
/*          LaserRENrhoR    The Radial unit direction of the laser */
/*          beam trajectory in the REN frame.                      */
/*          LaserRENrhoE    The East unit direction of the laser   */
/*          beam trajectory in the REN frame.                      */
/*          LaserRENrhoN    The North unit direction of the laser  */
/*          beam trajectory in the REN frame.                      */
/*          LaserRENrhoRDot The Radial unit velocity of the laser  */
/*          beam trajectory in the REN frame in unit dir*radians/sec */
/*          LaserRENrhoEDot The East unit velocity of the laser    */
/*          beam trajectory in the REN frame in unit dir*radians/sec */
/*          LaserRENrhoNDot The North unit velocity of the laser   */
/*          beam trajectory in the REN frame in unit dir*radians/sec */
/*          LaserRENrhoRDotDot The Radial unit accel. of the laser */
/*          beam trajectory in the REN frame in unit dir*radians/sec^2 */
/*          LaserRENrhoEDotDot The East unit accel. of the laser   */
/*          beam trajectory in the REN frame in unit dir*radians/sec^2 */
/*          LaserRENrhoNDotDot The North unit accel. of the laser  */

```

```

/*          lazer beam trajectory in the REN*/
/*          frame in unit dir*radians/sec^2 */
/*          SatRENrhoR      The Radial Component of the */
/*          position vector of the satellite*/
/*          wrt the platform in the REN      */
/*          coordinate frame.                */
/*          SatRENrhoE      The East Component of the */
/*          position vector of the satellite*/
/*          wrt the platform in the REN      */
/*          coordinate frame.                */
/*          SatRENrhoN      The North Component of the */
/*          position vector of the satellite*/
/*          wrt the platform in the REN      */
/*          coordinate frame.                */
/*          SatRENrhoRDot   The Radial Component of the */
/*          velocity vector of the satellite*/
/*          wrt the platform in the REN      */
/*          coordinate frame.                */
/*          SatRENrhoEDot   The East Component of the */
/*          velocity vector of the satellite*/
/*          wrt the platform in the REN      */
/*          coordinate frame.                */
/*          SatRENrhoNDot   The North Component of the */
/*          velocity vector of the satellite*/
/*          wrt the platform in the REN      */
/*          coordinate frame.                */
/*          SatRENrhoRDotDot The Radial Component of the */
/*          accel vector of the satellite */
/*          wrt the platform in the REN      */
/*          coordinate frame.                */
/*          SatRENrhoEDotDot The East Component of the */
/*          accel vector of the satellite */
/*          wrt the platform in the REN      */
/*          coordinate frame.                */
/*          SatRENrhoNDotDot The North Component of the */
/*          accel vector of the satellite */
/*          wrt the platform in the REN      */
/*          coordinate frame.                */
/*          */
/* OUTPUTS:      NAME:      DESCRIPTION: */
/*          SeparationAngle  The separation (in radians) of */
/*          the LaserRENrho and */
/*          PlatformSatRENrho vectors. */
/*          SeparationAngleDot The rate of change (in rad/sec) */
/*          of the separation of LaserRENrho*/
/*          PlatformSatRENrho vectors. */
/*          SeparationAngleDotDot The acceleration (in rad/sec^2) */
/*          of the separation of LaserRENrho*/
/*          and PlatformSatRENrho vectors. */
/*          ErrorList        The Errors which have occurred */
/*          */
/* COMPILER:      Borland C++ Builder3 Standard version */
/*          This compiler should be used to compile and link. */
/*          */
/*****
void FindSeparationAngle(double LaserRENrhoR,
                        double LaserRENrhoE,
                        double LaserRENrhoN,
                        double LaserRENrhoRDot,
                        double LaserRENrhoEDot,
                        double LaserRENrhoNDot,
                        double LaserRENrhoRDotDot,
                        double LaserRENrhoEDotDot,

```

```

double LaserRENrhoNDotDot,
double SatRENrhoR,
double SatRENrhoE,
double SatRENrhoN,
double SatRENrhoRDot,
double SatRENrhoEDot,
double SatRENrhoNDot,
double SatRENrhoRDotDot,
double SatRENrhoEDotDot,
double SatRENrhoNDotDot,
double &SeparationAngleInRadians,
double &SepAngleDot,
double &SepAngleDotDot,
ErrorStructure &ErrorList);

#endif

```

## C.6 The Process Satellite Library

The Process Satellite Library holds the modules responsible for tying together all of the modules mentioned previously to successfully evaluate a single satellite. This library is composed of four modules, ProcessSatellite, InterpolateVertex, FindDisplacementAnglesAgain, and TargetPlatformAgain. ProcessSatellite is the master module that uses all of the other modules to complete the evaluation of a single satellite. Process Satellite can be run independently of the Main Processor by using a calling program such as the GUI used in this project, shown in Figure C.7.

Aircraft Information		Satellite Information		OUTPUTS	
Aircraft Latitude Degrees	10 N	Satellite Number	0	Range (km)	0
Aircraft Latitude Minutes	0	Classification	0	Time (deg)	0
Aircraft Latitude Seconds	0	International ID	0	FORCAST SEPARATION ANGLE CRITERIA	
Aircraft Longitude Degrees	55	Epoch Year	0	Dist (deg)	0
Aircraft Longitude Minutes	0	Epoch Day	0	SepDist (deg/s)	0
Aircraft Longitude Seconds	0	Revs/Day Squared	0	SepDist (deg/s <sup>2</sup> )	0
Aircraft Altitude	129	Revs/Day Cubed	0	INTERPOLATION RESULTS	
Aircraft Xdot (ECEF) km/h	0	BStar Drag	0	Interpolation Used?	NO
Aircraft Ydot (ECEF) km/h	300	Ephemeris Type	0	Time until Intersection (sec)	0.0
Aircraft Zdot (ECEF) km/h	0	EI Set Number	0	Closest Approach Angle	0.0
INPUTS		Inclination	0	Will Launch Cross Satellite Path?	NO
Tune Az and Elevation in deg		Right Ascension	0	Greenwich Meridian Reference	
Laser Azimuth	45.0	Eccentricity	0	Reference Hour	4
Laser Elevation	45.0	Arg of Perigee	0	Reference Minute	40
Azimuth Dot	0.2	Mean Anomaly	0	Reference Second	1.299
Elevation Dot	1.6	Mean Motion	0	Modified Julian Date	11029.0
Azimuth Dot Dot	0.0001	Rev At Epoch	0	Greenwich Time of Laser Start	
Elevation Dot Dot	0.003	Populate Using File		Year	1998
Sat Pos Error (m)	10000	PAData/TLEfile5.txt		Month (1-12)	8
Plane Pos Error (m)	25	Process Satellite		Day	14
Missile Pos Error (m)	25	No Errors		Hour	3
Range to Missile (km)	200			Minute	58
Other Error Angles (deg)	0.0			Second	2.0
Interpolation Increment (sec)	0.1			Duration of Laser (sec)	30.0
Interval from Forecast Vertex	2.0				
(Recommend 2.0 sec interval with 0.1 Interpolation Increment)					

Figure C.7. GUI Used to Run and Test ProcessSatellite Module

### C.6.1 ProcessSatellite

As mentioned previously, ProcessSatellite is one of the master modules of the Main Processor. It is responsible for tying together all of the modules discussed so far, to process and evaluate a single satellite. It first calls FindDisplacementAngles to find a forecasted intercept time, and then, if the satellite is forecasted to intersect the laser, it calls InterpolateVertex to more closely examine that intersection point.

#### Inputs

**struct Satellite &Sat** The structure holding all of the satellite ephemeris information. The Type "Satellite" is defined in Satellite.h.

**struct Aircraft &ABLPlatform** ABLPlatform is a structure of type "Aircraft" that holds all of the information about the position of the aircraft at the time of execution of the Preprocessor.

**double JulianDate** This is the modified Julian Date (The Julian Date – 2440000) that needs to be propagated to. This is the actual time at which the user wishes to find the position of the satellite.

**double AzimuthInDegrees** The current azimuth of the laser turret in degrees.

**double ElevationInDegrees** The current elevation of the laser turret in degrees.

**double AzimuthDot** The current rate of change of the azimuth of the laser turret in degrees per second.

**double ElevationDot** The current rate of change of the elevation of the laser turret in degrees per second.

**double AzimuthDotDot** The current acceleration of the azimuth of the laser turret in degrees per second.

**double ElevationDotDot** The current acceleration of the elevation of the laser turret in degrees per second.

**double SatPositionErrorInMeters** The radius of the "sphere" inside which the satellite is known to reside (in meters) throughout the forecast period

(around 10 to 30 seconds). For SGP4, this may be as high as around 10 km (10000 m).

**double PlatformPositionErrorInMeters** The radius of the “sphere” inside which the platform is known to reside (in meters) throughout the forecast period (around 10 to 30 seconds). For a 747 on autopilot with GPS tracking, a rough estimate might be 50 meters.

**double MissilePositionErrorInMeters** The radius of the “sphere” inside which the missile is known to reside (in meters) throughout the forecast period (around 10 to 30 seconds). This may just be an educated guess. It is difficult to now the behavior of the missile based on initial conditions, and this parameter will have to be given some thought.

**double RangeToMissileInKilometers** The range from the platform to the missile.

**double OtherErrorAngleInDeg** This is a “catch-all” parameter, to be used if, in the future, there are other error angles that crop up that have not already been accounted for.

**int ReferenceHour** Reference hour Refers to the Reference angle of  $\theta_g$  (The angle between the Greenwich meridian and the Vernal Equinox). This angle is given in hours, minutes and seconds as opposed to degrees or radians. This parameter holds the hours portion of  $\theta_g$ .

**int ReferenceMinute** The minutes portion of  $\theta_g$ .

**double ReferenceSecond** The seconds portion of  $\theta_g$ .

**double RefModJulianDate** This parameter holds the Modified Julian Date at which the reference angle,  $\theta_g$ , was taken. This allows  $\theta_g$  to be propagated forward to the present moment.

**double SecondsFromVertex** This input parameter describes the number of seconds before the forecast intercept time the the user desires to analyze via interpolation. For a better explanation on interpolation, see Chapter III. The author recommends this time interval be around 2.0 seconds.

**double InterpolationIncrement** The amount of time that transpires between samplings when interpolating the vertex. For a better extplanation on interpolation, see Chapter III. The author recommends this time interval be around 0.1 seconds.

**double LazeDuration** The expected amount of time that the lazer will be "on", or illuminating its target. It is estimated that this value should never operationally exceed thirty seconds.

## Outputs

**double ThetaGInRad** This is the angle at which the Earth's Greenwich Meridian is currently at with respect to the ECI frame, where the referent angle is the Vernal Equinox. This angle should be in radians.

**double &RangeInKilometers** Range to the satellite from the platform in kilometers.

**double &ErrorAngleInRadians** The final error angle (in radians) resulting from the position errors given previously.

**double &SeparationAngle** The separation angle between the laser turret unit position vector and the satellite position vector in the REN frame.

**double &SepAngleDot** The rate of change of the separation angle in radians per second.

**double &SepAngleDotDot** The acceleration of the separation angle in radians per second squared.

**int &Intersection** This is a boolean value that informs the user as to whether an intersection has been determined to occur or not.

0 = No intersection determined

1 = Intersection determined

**int &Interpolation** This is a boolean parameter that tells the user whether or not the satellite was forecast to come close enough to the laser vector to warrant an interpolation.

0 = Satellite not close enough to warrant interpolation.

1 = Satellite approached close enough, interpolation performed.

**double &TimeToIntersect** If an intersection is determined to occur, this parameter will tell how many seconds into the future this intersection will occur. If an intersection does not occur, this parameter will be set to 0.0.

**double &ClosestApproachInDegrees** The closest that this satellite ever comes to the laser position vector.

**ErrorStructure &ErrorList** This is the error-handling structure that is used as both an input (to see if errors have occurred) and an output (to list any errors occurring in this module).



### **C.6.2 InterpolateVertex**

InterpolateVertex is responsible for propagation of the actual positions of the different players of the satellite analysis in time. This actual propagation is used to find the “real” separation angle between the satellite position vector and the laser position vector at different points in time. Although this procedure yields a fairly exact forecast for the separation angle, it is not used for every satellite, because it is calculation intensive. Rather ProcessSatellite uses a rough forecast based upon the initial separation angle, the rate of change, and the acceleration of the angle. If this initial forecast indicates a close-approach (an intersection, according to the forecast) of the satellite, then and only then is InterpolateVertex called to fully evaluate that close approach. Because many of the calculations needed to propagate positions in time have already been done, InterpolateVertex does not call FindDisplacementAngles or TargetPlatform, as might normally be expected when trying to find the new position of the platform and the separation angle. Rather, a shortened version of these two modules, FindDisplacementAnglesAgain, and TargetPlatformAgain, were created to lighten the processing load. Further examination of these modules will reveal that even more can be stripped from these modules to further condense the amount of processing needed. Unfortunately, there was not enough time to “optimize” these two modules and perform the necessary testing again, as of the writing of this final draft. However, it should not be that difficult to identify the portions of these shortened modules that are not needed to find only the separation angle. As the software stands, however, it is still running within an acceptable time margin. InterpolateVertex is the only module to call FindDisplacementAnglesAgain and TargetPlatformAgain.

## Inputs

**struct Satellite &Sat** The structure holding all of the satellite ephemeris information. The Type "Satellite" is defined in Satellite.h.

**struct Aircraft &ABLPlatform** ABLPlatform is a structure of type "Aircraft" that holds all of the information about the position of the aircraft at the time of execution of the Preprocessor.

**int ReferenceHour** Reference hour Refers to the Reference angle of  $\theta_g$  (The angle between the Greenwich meridian and the Vernal Equinox). This angles is given in hours, minutes and seconds as opposed to degrees or radians. This parameter holds the hours portion of  $\theta_g$ .

**int ReferenceMinute** The minutes portion of  $\theta_g$ .

**double ReferenceSecond** The seconds portion of  $\theta_g$ .

**double RefModJulianDate** This parameter holds the Modified Julian Date at which the reference angle,  $\theta_g$ , was taken. This allows  $\theta_g$  to be propagated forward to the present moment.

**double LazeDuration** The expected amount of time that the lazer will be "on", or illuminating its target. It is estimated that this value should never operationally exceed thirty seconds.

**double JulianDate** This is the modified Julian Date (The Julian Date – 2440000) that needs to be propagated to. This is the actual time at which the user wishes to find the position of the satellite.

**double AzimuthInDegrees** The current azimuth of the laser turret in degrees.

**double ElevationInDegrees** The current elevation of the laser turret in degrees.

**double AzimuthDot** The current rate of change of the azimuth of the laser turret in degrees per second.

**double ElevationDot** The current rate of change of the elevation of the laser turret in degrees per second.

**double AzimuthDotDot** The current acceleration of the azimuth of the laser turret in degrees per second.

**double ElevationDotDot** The current acceleration of the elevation of the laser turret in degrees per second.

**double ErrorAngleInRadians** The final error angle (in radians) resulting from the position errors given previously.

**double SecondsFromVertex** This input parameter describes the number of seconds before the forecast intercept time the user desires to analyze via interpolation. For a better explanation on interpolation, see Chapter III. The author recommends this time interval be around 2.0 seconds.

**double InterpolationIncrement** The amount of time that transpires between samplings when interpolating the vertex. For a better explanation on interpolation, see Chapter III. The author recommends this time interval be around 0.1 seconds.

### Outputs

**double &TimeToIntersect** If an intersection is determined to occur, this parameter will tell how many seconds into the future this intersection will occur. If an intersection does not occur, this parameter will be set to 0.0.

**double &ClosestApproachInDegrees** The closest that this satellite ever comes to the laser position vector.

**ErrorStructure      &ErrorList** This is the error-handling structure that is used as both an input (to see if errors have occurred) and an output (to list any errors occurring in this module).

### **C.6.3 FindDisplacementAnglesAgain**

FindDisplacementAnglesAgain is a module that is a shortened version of the original FindDisplacementAngles module that found the separation angle, as well as the rate of change and acceleration of that angle. To prevent azimuth and elevation reconversion to ECI coordinates, three new parameters were added to simply propagate the platforms motion in the ECEF frame. Admittedly, this module could have been done more thoughtfully. Time was running out, and a quick fix was needed to make

InterpolateVertex run. Further optimization efforts should start here, with these two modules.

### Inputs

**struct Satellite &Sat** The structure holding all of the satellite ephemeris information. The Type "Satellite" is defined in Satellite.h.

**struct Aircraft &ABLPlatform** ABLPlatform is a structure of type "Aircraft" that holds all of the information about the position of the aircraft at the time of execution of the Preprocessor.

**double ThetaGInRad** This is the angle at which the Earth's Greenwich Meridian is currently at with respect to the ECI frame, where the referent angle is the Vernal Equinox. This angle should be in radians.

**double JulianDate** This is the modified Julian Date (The Julian Date – 2440000) that needs to be propagated to. This is the actual time at which the user wishes to find the position of the satellite.

**double ChangeInX** This is the ECEF X propagation distance used to propagate the platform position forward in time.

**double ChangeInY** This is the ECEF Y propagation distance used to propagate the platform position forward in time.

**double ChangeInZ** This is the ECEF Z propagation distance used to propagate the platform position forward in time.

**double AzimuthInDegrees** The current azimuth of the laser turret in degrees.

**double ElevationInDegrees** The current elevation of the laser turret in degrees.

**double AzimuthDot** The current rate of change of the azimuth of the laser turret in degrees per second.

**double ElevationDot** The current rate of change of the elevation of the laser turret in degrees per second.

**double AzimuthDotDot** The current acceleration of the azimuth of the laser turret in degrees per second.

**double ElevationDotDot** The current acceleration of the elevation of the laser turret in degrees per second.

**double ErrorAngleInRadians** The final error angle (in radians) resulting from the position errors given previously. In the Original FindDisplacementAngles, this was an output parameter, but since it has already been found, it need not be recalculated for the brief forecast time.

## Outputs

**double &LaserRENrhoR** The current REN R (Radial) unit position vector of the position of the laser with respect to the platform, as derived from the azimuth and elevation.

**double &LaserRENrhoE** The current REN E (East) unit position vector of the position of the laser with respect to the platform, as derived from the azimuth and elevation.

**double &LaserRENrhoN** The current REN N (North) unit position vector of the position of the laser with respect to the platform, as derived from the azimuth and elevation

**double &LaserRENrhoRDot** The current REN R (Radial) velocity vector of the velocity of the laser with respect to the platform, as derived from the azimuth, elevation and the rate of change of the azimuth and elevation.

**double &LaserRENrhoEDot** The current REN E (East) velocity vector of the velocity of the laser with respect to the platform, as derived from the azimuth, elevation and the rate of change of the azimuth and elevation.

**double &LaserRENrhoNDot** The current REN N (North) velocity vector of the velocity of the laser with respect to the platform, as derived from the azimuth, elevation and the rate of change of the azimuth and elevation.

**double &LaserRENrhoRDotDot** The current REN R (Radial) acceleration vector of the acceleration of the laser with respect to the platform, as derived from the azimuth, elevation, the rate of change of the azimuth and elevation, and the acceleration of the azimuth and elevation.

**double &LaserRENrhoEDotDot** The current REN E (East) acceleration vector of the acceleration of the laser with respect to the platform, as derived from the azimuth, elevation, the rate of change of the azimuth and elevation, and the acceleration of the azimuth and elevation.

**double &LaserRENrhoNDotDot** The current REN N (North) acceleration vector of the acceleration of the laser with respect to the platform, as derived from

the azimuth, elevation, the rate of change of the azimuth and elevation, and the acceleration of the azimuth and elevation.

**double &RangeToSatInKilometers** Range to the satellite from the platform in kilometers.

**double &SeparationAngle** The separation angle between the laser turret unit position vector and the satellite position vector in the REN frame.

**double &SepAngleDot** The rate of change of the separation angle in radians per second.

**double &SepAngleDotDot** The acceleration of the separation angle in radians per second squared.

**ErrorStructure &ErrorList** This is the error-handling structure that is used as both an input (to see if errors have occurred) and an output (to list any errors occurring in this module).

#### **C.6.4 TargetPlatformAgain**

TargetPlatformAgain is a shortened version of TargetPlatform that uses a set of platform position propagation parameters, ChangeInX, ChangeInY, and ChangeInZ to propagate the platform position forward in time, rather than recomputing position from latitude and longitude.

#### **Inputs**

**struct Aircraft &ABLPlatform** ABLPlatform is a structure of type "Aircraft" that holds all of the information about the position of the aircraft at the time of execution of the Preprocessor.

**double ThetaGInRad** This is the angle at which the Earth's Greenwich Meridian is currently at with respect to the ECI frame, where the referent angle is the Vernal Equinox. This angle should be in radians.

**double JulianDate** This is the modified Julian Date (The Julian Date – 2440000) that needs to be propagated to. This is the actual time at which the user wishes to find the position of the satellite.

**double ChangeInX** This is the ECEF X propagation distance used to propagate the platform position forward in time.

**double ChangeInY** This is the ECEF Y propagation distance used to propagate the platform position forward in time.

**double ChangeInZ** This is the ECEF Z propagation distance used to propagate the platform position forward in time.

## Outputs

**double &PlatformECIRhoX** The current ECI X position vector of the position of the platform with respect to the center of the Earth, as derived from the latitude and longitude given in the Aircraft structure input.

**double &PlatformECIRhoY** The current ECI Y position vector of the position of the platform with respect to the center of the Earth, as derived from the latitude and longitude given in the Aircraft structure input.

**double &PlatformECIRhoZ** The current ECI Z position vector of the position of the platform with respect to the center of the Earth, as derived from the latitude and longitude given in the Aircraft structure input.

**double &PlatformECIRhoXDot** The current ECI X velocity vector of the velocity of the platform with respect to the center of the Earth, as derived from the ECEF velocities given in the Aircraft structure input.

**double &PlatformECIRhoYDot** The current ECI Y velocity vector of the velocity of the platform with respect to the center of the Earth, as derived from the ECEF velocities given in the Aircraft structure input.

**double &PlatformECIRhoZDot** The current ECI Z velocity vector of the velocity of the platform with respect to the center of the Earth, as derived from the ECEF velocities given in the Aircraft structure input.

**double &PlatformECIRhoXDotDot** The current ECI X acceleration vector of the acceleration of the platform with respect to the center of the Earth, as derived from the centripetal and Coriolis accelerations only. It is assumed that the aircraft itself is maintaining straight and level flight at constant velocity.

**double &PlatformECIRhoYDotDot** The current ECI Y acceleration vector of the acceleration of the platform with respect to the center of the Earth, as derived from the centripetal and Coriolis accelerations only. It is assumed that the aircraft itself is maintaining straight and level flight at constant velocity.

**double &PlatformECIRhoZDotDot** The current ECI Z acceleration vector of the acceleration of the platform with respect to the center of the Earth, as derived from the centripetal and Coriolis accelerations only. It is assumed that the aircraft itself is maintaining straight and level flight at constant velocity.

**double &PlatformRENrhoR** The current REN R (Radial) position vector of the position of the platform with respect to the center of the Earth, as derived from the ECI position vector, rotated into the REN frame.

**double &PlatformRENrhoE** The current REN E (East) position vector of the position of the platform with respect to the center of the Earth, as derived from the ECI position vector, rotated into the REN frame.

**double &PlatformRENrhoN** The current REN N (North) position vector of the position of the platform with respect to the center of the Earth, as derived from the ECI position vector, rotated into the REN frame.

**double &PlatformRENrhoRDot** The current REN R (Radial) velocity vector of the velocity of the platform with respect to the center of the Earth, as derived from the ECI velocity vector, rotated into the REN frame.

**double &PlatformRENrhoEDot** The current REN E (East) velocity vector of the velocity of the platform with respect to the center of the Earth, as derived from the ECI velocity vector, rotated into the REN frame.

**double &PlatformRENrhoNDot** The current REN N (North) velocity vector of the velocity of the platform with respect to the center of the Earth, as derived from the ECI velocity vector, rotated into the REN frame.

**double &PlatformRENrhoRDotDot** The current REN R acceleration vector of the acceleration of the platform with respect to the center of the Earth, as derived from the centripetal and Coriolis accelerations derived in the ECI frame and rotated into the REN frame. It is assumed that the aircraft itself is maintaining straight and level flight at constant velocity.

**double &PlatformRENrhoEDotDot** The current REN E acceleration vector of the acceleration of the platform with respect to the center of the Earth, as derived from the centripetal and Coriolis accelerations derived in the ECI frame and rotated into the REN frame. It is assumed that the aircraft itself is maintaining straight and level flight at constant velocity.

**double &PlatformRENrhoNDotDot** The current REN N acceleration vector of the acceleration of the platform with respect to the center of the Earth, as derived from the centripetal and Coriolis accelerations derived in the ECI frame and rotated into the REN frame. It is assumed that the aircraft itself is maintaining straight and level flight at constant velocity.



**double &ECItORENMatrix11** This an element (row 1, column 1) of the matrix used to transform a vector from the ECI coordinate frame to the REN coordinate frame.

**double &ECItORENMatrix12** This an element (row 1, column 2) of the matrix used to transform a vector from the ECI coordinate frame to the REN coordinate frame.

**double &ECItORENMatrix13** This an element (row 1, column 3) of the matrix used to transform a vector from the ECI coordinate frame to the REN coordinate frame.

**double &ECItORENMatrix21** This an element (row 2, column 1) of the matrix used to transform a vector from the ECI coordinate frame to the REN coordinate frame.

**double &ECItORENMatrix22** This an element (row 2, column 2) of the matrix used to transform a vector from the ECI coordinate frame to the REN coordinate frame.

**double &ECItORENMatrix23** This an element (row 2, column 3) of the matrix used to transform a vector from the ECI coordinate frame to the REN coordinate frame.

**double &ECItORENMatrix31** This an element (row 3, column 1) of the matrix used to transform a vector from the ECI coordinate frame to the REN coordinate frame.

**double &ECItORENMatrix32** This an element (row 3, column 2) of the matrix used to transform a vector from the ECI coordinate frame to the REN coordinate frame.

**double &ECItORENMatrix33** This an element (row 3, column 3) of the matrix used to transform a vector from the ECI coordinate frame to the REN coordinate frame.

**ErrorStructure      &ErrorList** This is the error-handling structure that is used as both an input (to see if errors have occurred) and an output (to list any errors occurring in this module).

## C.6.5 The TargetSatellite.h Header File

```

#ifndef ProcessSatelliteH
#define ProcessSatelliteH
/*****
/*  MODULE NAME:      ProcessSatellite.h                      */
/*  AUTHOR:          Captain David Vloedman                  */
/*  DATE CREATED:    14 January, 1999                        */
/*                                                           */
/*  PURPOSE:         This module supports the meat of the Main Processor and */
/*                   is used to evaluate the error angle and the displacement*/
/*                   angle between the laser position vector in the REN frame*/
/*                   and the satellite position vector in the same frame. It*/
/*                   uses this angle and its rate of change to determine when*/
/*                   and if the satellite will intersect the path of the    */
/*                   laser.                                                  */
/*                                                           */
/*  COMPILER:        Borland C++ Builder3 Standard version    */
/*                   This compiler should be used to compile and link.      */
/*                                                           */
*****/
/*****
/* C++BUILDER-SPECIFIC LIBRARIES */
*****/
#include <vcl.h>
#pragma hdrstop
#pragma package(smart_init)
/*****
/* USER-BUILT LIBRARIES          */
*****/
#include "TimeModules.h"
#include "TLEInput.h"
#include "LaserConstants.h"
#include "Satellite.h"
#include "Aircraft.h"
#include "ErrorStructure.h"
#include "EvaluateEphemerisModules.h"
#include "SGP4SupportModules.h"
#include "FindDisplacementAngleModules.h"
#include "TargetSatellite.h"
#include "TargetPlatform.h"
#include "TargetLaser.h"
/*****
/* C STANDARD LIBRARIES          */
*****/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <iostream.h>
#include <conio.h>
#include <math.h>

/*****
/* ***** FUCTIONS ***** */
*****/

/*****
/*  FUNCTION NAME:      ProcessSatellite                      */
/*  AUTHOR:          Captain David Vloedman                  */
/*  DATE CREATED:    January 13, 1999                        */
/*                                                           */
/*  PURPOSE:         This module supports the meat of theMain Processor and */

```

```

/*          is used to evaluate the error angle and the displacement*/
/*          angle between the laser position vector in the REN frame*/
/*          and the satellite position vector in the same frame. It*/
/*          uses this angle and its rate of change to determine when*/
/*          and if the satellite will intersect the path of the    */
/*          laser.                                                  */
/*                                                                  */
/*                                                                  */
/*          COMPILER:      Borland C++ Builder3 Standard version    */
/*          This compiler should be used to compile and link.      */
/*                                                                  */
/*****
void ProcessSatellite(struct Aircraft &Platform,
                      struct Satellite &Sat,
                      int    ReferenceHour,
                      int    ReferenceMinute,
                      double ReferenceSecond,
                      double RefModJulianDate,
                      double SecondsFromVertex,
                      double InterpolationIncrement,
                      double &ThetaGInRad,
                      double JulianDate,
                      double LazeDuration,
                      double LaserAzimuthInDegrees,
                      double LaserAzimuthDot,
                      double LaserAzimuthDotDot,
                      double LaserElevationInDegrees,
                      double LaserElevationDot,
                      double LaserElevationDotDot,
                      double SatPositionErrorInMeters,
                      double PlatformPositionErrorInMeters,
                      double MissilePositionErrorInMeters,
                      double RangeToMissileInKilometers,
                      double OtherErrorAngleInDeg,
                      double &RangeInKilometers,
                      double &ErrorAngleInRadians,
                      double &SeparationAngle,
                      double &SepAngleDot,
                      double &SepAngleDotDot,
                      int    &Intersection,
                      int    &Interpolation,
                      double &TimeToIntersect,
                      double &ClosestApproachInDegrees,
                      ErrorStructure &ErrorList);

/*****
/*  FUNCTION NAME:  InterpolateVertex                                */
/*  AUTHOR:        Captain David Vloedman                          */
/*  DATE CREATED:   January 13, 1999                                */
/*  PURPOSE:        This module supports the meat of the Main Processor and */
/*                  is used to evaluate the error angle and the displacement*/
/*                  angle between the laser position vector in the REN frame*/
/*                  and the satellite position vector in the same frame    */
/*                  during the relatively short time of estimated closest   */
/*                  approach of the two vectors. The smaller the inter-    */
/*                  polation increment, the more accurate the estimate, and */
/*                  the longer the processing time.                        */
/*  COMPILER:      Borland C++ Builder3 Standard version            */
/*  This compiler should be used to compile and link.                */

```

```

/*****
void InterpolateVertex(struct Aircraft &Platform,
                      struct Satellite &Sat,
                      int    ReferenceHour,
                      int    ReferenceMinute,
                      double ReferenceSecond,
                      double RefModJulianDate,
                      double JulianDate,
                      double LazeDuration,
                      double LaserAzimuthInDegrees,
                      double LaserAzimuthDot,
                      double LaserAzimuthDotDot,
                      double LaserElevationInDegrees,
                      double LaserElevationDot,
                      double LaserElevationDotDot,
                      double ErrorAngleInRadians,
                      double SecondsFromVertex,
                      double InterpolationIncrement,
                      double &TimeToIntersect,
                      double &ClosestApproachInDegrees,
                      ErrorStructure &ErrorList);

/*****
/*  FUNCTION NAME:  TargetPlatformAgain
/*  AUTHOR:        Captain David Vloedman
/*  DATE CREATED:  January 24, 1998
/*
/*  PURPOSE:       This function will take the position of the aircraft and*
/*                  position, velocity and acceleration in the REN frame of *
/*                  the Airborn laser platform. This is very similar to *
/*                  "TargetPlatform", but uses slightly different input *
/*                  parameters.
/*
/*  COMPILER:      Borland C++ Builder3 Standard version
/*                  This compiler should be used to compile and link.
/*
/*****
void TargetPlatformAgain(struct Aircraft &Platform,
                        double &ThetaGInRad,
                        double JulianDate,
                        double ChangeInX,
                        double ChangeInY,
                        double ChangeInZ,
                        double &PlatformECIRhoX,
                        double &PlatformECIRhoY,
                        double &PlatformECIRhoZ,
                        double &PlatformECIRhoXDot,
                        double &PlatformECIRhoYDot,
                        double &PlatformECIRhoZDot,
                        double &PlatformECIRhoXDotDot,
                        double &PlatformECIRhoYDotDot,
                        double &PlatformECIRhoZDotDot,
                        double &PlatformRENrhoR,
                        double &PlatformRENrhoE,
                        double &PlatformRENrhoN,
                        double &PlatformRENrhoRDot,
                        double &PlatformRENrhoEDot,
                        double &PlatformRENrhoNDot,
                        double &PlatformRENrhoRDotDot,
                        double &PlatformRENrhoEDotDot,
                        double &PlatformRENrhoNDotDot,
                        double &ECitoRENMatrix11,

```

```

        double &ECitoRENMatrix12,
        double &ECitoRENMatrix13,
        double &ECitoRENMatrix21,
        double &ECitoRENMatrix22,
        double &ECitoRENMatrix23,
        double &ECitoRENMatrix31,
        double &ECitoRENMatrix32,
        double &ECitoRENMatrix33,
        ErrorStructure &ErrorList);

/*****
/*  FUNCTION NAME:  FindDisplacementAnglesAgain
/*  AUTHOR:        Captain David Vloedman
/*  DATE CREATED:  January 23, 1999
/*
/*  PURPOSE:       This function will take satellite and platform data and
/*                  will use it to find the error angle and the displacement
/*                  angle between the laser position vector in the REN frame
/*                  and the satellite position vector in the same frame.
/*                  NOTICE THAT THIS IS NOT "FindDisplacementAngles", BUT
/*                  "FindDisplacementAnglesAgain". IT IS ONLY SLIGHTLY
/*                  DIFFERENT THAN THE OTHER, INCORPORATING THE THREE INPUT
/*                  PARAMETERS ChangeInX, ChangeInY AND ChangeInZ WHICH
/*                  DESCRIBES A SLIGHT POSITION CHANGE IN THE ECEF FRAME.
/*
/*  COMPILER:      Borland C++ Builder3 Standard version
/*                  This compiler should be used to compile and link.
/*
*****/
void FindDisplacementAnglesAgain(struct Aircraft &Platform,
                                struct Satellite &Sat,
                                double &ThetaGInRad,
                                double JulianDate,
                                double ChangeInX,
                                double ChangeInY,
                                double ChangeInZ,
                                double LaserAzimuthInDegrees,
                                double LaserAzimuthDot,
                                double LaserAzimuthDotDot,
                                double LaserElevationInDegrees,
                                double LaserElevationDot,
                                double LaserElevationDotDot,
                                double &PlatformSatRENrhoR,
                                double &PlatformSatRENrhoE,
                                double &PlatformSatRENrhoN,
                                double &PlatformSatRENrhoRDot,
                                double &PlatformSatRENrhoEDot,
                                double &PlatformSatRENrhoNDot,
                                double &PlatformSatRENrhoRDotDot,
                                double &PlatformSatRENrhoEDotDot,
                                double &PlatformSatRENrhoNDotDot,
                                double &LaserRENrhoR,
                                double &LaserRENrhoE,
                                double &LaserRENrhoN,
                                double &LaserRENrhoRDot,
                                double &LaserRENrhoEDot,
                                double &LaserRENrhoNDot,
                                double &LaserRENrhoRDotDot,
                                double &LaserRENrhoEDotDot,
                                double &LaserRENrhoNDotDot,
                                double &RangeToSatInKilometers,
                                double &ErrorAngleInRadians,

```

```
double &SeparationAngle,  
double &SepAngleDot,  
double &SepAngleDotDot,  
ErrorStructure &ErrorList);  
  
#endif
```

## C.7 The ABLPA Main Processor

The Main Processor is the final culmination of all of the software discussed previously. The task of the Main Processor is to take the output file containing all active satellites in view of the platform, and create two output files. The first output file will contain all of the satellites that are forecast to be intersected by the laser during the laser duration. The Processor also creates an output file containing the satellites that pass closely enough to the laser path to be “interpolated”, or analyzed, but may or may not actually intersect the laser. This second “close-approach” file is used more for testing and verification than operational use. Statistically, more often than not the Intersection File will be empty after

The graphical interface is divided into several sections for data entry and output:

- Aircraft Information:** Fields for Latitude (10 N), Longitude (55), Altitude (12.9), and various coordinates (X, Y, Z).
- Greenwich Meridian Reference:** Fields for Hour (4), Minute (40), Second (1.299), and Modified Julian Date (11029.0).
- Inputs:** Fields for Laser Azimuth (45.0), Elevation (45.0), Range (0.2), Elevation Rate (1.6), Azimuth Rate (0.0001), Elevation Rate Rate (0.003), and various time and range parameters.
- Simulation Time:** Fields for Year (1998), Month (8), Day (14), Hour (3), Minute (58), Second (2.0), and Duration (30).
- Results:** Fields for Number of Satellites Evaluated (0), Number of Satellites Intersected (0), and Number of Satellites Interpolated (0).
- Output Files:** Fields for Preprocessor Output File (PAData/PreprocessorOutput.txt), Intersected Satellite File (PAData/IntersectingSatelliteOutput.txt), and Closest Approach File (PAData/ClosestApproachOutput.txt).

**Figure C.8. The Graphical Interface Developed to Run the Main Processor**

a full Main Processor run-through, because the chances of “hitting” a satellite (even with a theoretical error-angle) is slim. This close approach file can be used to verify the successful run and processing of the Main Processor. Both output files have exactly the same format as the main TLE file, except they have fewer (or no) satellites within them. This processor can be run independently, by a calling module made by the user, or by using the GUI developed to run and test the processor during project development. This graphical interface is illustrated in Figure C.8.

### **C.7.1 PAMainProcessor**

The Main Processor calls the ReadTLEFile module to accomplish loading of all of the satellites in the input file, and then makes repeated calls to ProcessSatellite to analyze each satellite in turn. It can be seen that the format of the Main Processor is very similar to the Preprocessor, using input and output TLE files to accomplish most of the satellite information transfer.

### **C.7.2 Inputs**

**char InFileName[MAXNAMELENGTH]** This parameter holds the name of the Two Line Element Set that holds the satellites to be evaluated.

**char OutFileName[MAXNAMELENGTH]** Holds the name of the file to which the intersected satellites’ Two-Line Element set information is routed to. This file holds all of the satellites that have been judged by the Processor to be intersected by the laser during the laser firing time.

**char ClosestApproachFileName[MAXNAMELENGTH]** Holds the name of the file to which the close-approach satellites’ Two-Line Element set information is routed to. This file holds all of the satellites that have been judged by the Processor to be close enough to the laser beam that interpolation is required.



**struct Aircraft &ABLPlatform** ABLPlatform is a structure of type "Aircraft" that holds all of the information about the position of the aircraft at the time of execution of the Processor.

**int ReferenceHour** Reference hour Refers to the Reference angle of  $\theta_g$  (The angle between the Greenwich meridian and the Vernal Equinox). This angles is given in hours, minutes and seconds as opposed to degrees or radians. This parameter holds the hours portion of  $\theta_g$ .

**int ReferenceMinute** The minutes portion of  $\theta_g$ .

**double ReferenceSecond** The seconds portion of  $\theta_g$ .

**double RefModJulianDate** This parameter holds the Modified Julian Date at which the reference angle,  $\theta_g$ , was taken. This allows  $\theta_g$  to be propagated forward to the present moment.

**int CalcYear** The current year.

**int CalcMonth** The current month (1-12).

**int CalcDay** The current day (1-31).

**int CalcHour** The current hour (1-24).

**int CalcMinute** The current minute (1-60).

**double CalcSecond** The current second. This is the only part of the current time that can be given as a non-integer. This field should be accurate to at least three decimal places.

**double LazeDuration** The expected amount of time that the lazer will be "on", or illuminating its target. It is estimated that this value should never operationally exceed thirty seconds.

**double LaserAzimuthInDegrees** The current azimuth of the laser turret in degrees.

**double LaserAzimuthDot** The current rate of change of the azimuth of the laser turret in degrees per second.

**double LaserAzimuthDotDot** The current acceleration of the azimuth of the laser turret in degrees per second.

**double LaserElevationInDegrees** The current elevation of the laser turret in degrees.

**double LaserElevationDot** The current rate of change of the elevation of the laser turret in degrees per second.

**double LaserAzimuthDotDot** The current acceleration of the azimuth of the laser turret in degrees per second.

**double LaserElevationDotDot** The current acceleration of the elevation of the laser turret in degrees per second.

**double SatPositionErrorInMeters** The radius of the “sphere” inside which the satellite is known to reside (in meters) throughout the forecast period (around 10 to 30 seconds). For SGP4, this may be as high as around 10 km (10000 m).

**double PlatformPositionErrorInMeters** The radius of the “sphere” inside which the platform is known to reside (in meters) throughout the forecast period (around 10 to 30 seconds). For a 747 on autopilot with GPS tracking, a rough estimate might be 50 meters.

**double MissilePositionErrorInMeters** The radius of the “sphere” inside which the missile is known to reside (in meters) throughout the forecast period (around 10 to 30 seconds). This may just be an educated guess. It is difficult to now the behavior of the missile based on initial conditions, and this parameter will have to be given some thought.

**double RangeToMissileInKilometers** The range from the platform to the missile.

**double OtherErrorAngleInDeg** This is a “catch-all” parameter, to be used if, in the future, there are other error angles that crop up that have not already been accounted for.

**double SecondsFromVertex** This input parameter describes the number of seconds before the forecast intercept time the the user desires to analyze via interpolation. For a better explanation on interpolation, see Chapter III. The author recommends this time interval be around 2.0 seconds.

**double InterpolationIncrement** The amount of time that transpires between samplings when interpolating the vertex. For a better explanation on interpolation, see Chapter III. The author recommends this time interval be around 0.1 seconds.

### C.7.3 Outputs

**int            &InFileLength** This parameter tells the user how many elements were read in from the file specified by the input parameter "InFileName[MAXNAMELENGTH]". This is the total number of satellites that will be evaluated during the run of the Processor.

**int            &OutFileLength** This parameter tells the user how many elements were written to the file specified by the input parameter "OutFileName[MAXNAMELENGTH]". This is the total number of satellites that were judged to be "intersected" of the laser between the time of the start of lazing and the end of the laze.

**int            &CloseApproachFileLength** This parameter tells the user how many elements were written to the file specified by the input parameter "CloseApproachFileName[MAXNAMELENGTH]". This should always be bigger than or equal to OutFileLength.

**double    &ThetaGInDegrees** This is the instantaneous angle between the Greenwich meridian and the Vernal Equinox at the moment of execution of the Processor.

**ErrorStructure    &ErrorList** This parameter is both an input and output parameter. Each module uses it to assess whether a fatal error has occurred somewhere else in the program, and uses it to record errors which may be important to the user.

## C.7.4 The PAMainProcessor.h Header File

```

#ifndef PAMainProcessorH
#define PAMainProcessorH
/*****
/*  MODULE NAME:      PAMainProcessor.h                */
/*  AUTHOR:          Captain David Vloedman            */
/*  DATE CREATED:    January 10, 1998                  */
/*                                                         */
/*  PURPOSE:         This module is the model of the Airborne Laser */
/*                   Predictive Avoidance Processor which may be used to */
/*                   determine whether or not a given Laser trajectory will */
/*                   intersect with any of a list of satellites fed to it. */
/*                                                         */
/*  COMPILER:        Borland C++ Builder3 Standard version */
/*                   This compiler should be used to compile and link. */
/*                                                         */
*****/
/*****
/* C++BUILDER-SPECIFIC LIBRARIES */
*****/
#include <vcl.h>
#pragma hdrstop
#pragma package(smart_init)
/*****
/* USER-BUILT LIBRARIES */
*****/
#include "TimeModules.h"
#include "TLEInput.h"
#include "LaserConstants.h"
#include "Satellite.h"
#include "Aircraft.h"
#include "ErrorStructure.h"
#include "EvaluateEphemerisModules.h"
#include "PAMainProcessor.h"
#include "SGP4SupportModules.h"
#include "FindDisplacementAngleModules.h"
#include "TargetSatellite.h"
#include "TargetPlatform.h"
#include "TargetLaser.h"
#include "ProcessSatellite.h"
/*****
/* C STANDARD LIBRARIES */
*****/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <iostream.h>
#include <conio.h>
#include <math.h>

/*****
/* ***** FUCTIONS ***** */
*****/

/*****
/*  FUNCTION NAME:    PAMainProcessor                */
/*  AUTHOR:          Captain David Vloedman            */
/*  DATE CREATED:    January 15, 1998                  */
/*                                                         */
*****/

```

```

/* PURPOSE:      This procedure will read in an input file of Two Line  */
/*               Element (TLE) sets and perform an analysis to determine */
/*               whether or not satellites will be intercepted by the     */
/*               path of the airborne platform laser.                    */
/*               */
/* INPUTS:      NAME:      DEFINITION:      */
/*               InFileName  Holds name of the satellite file*/
/*               OutFileName File that holds the sats that  */
/*               are forecasted by the software  */
/*               to be intercepted bt the laser. */
/*               ClosestApproachFileName File that holds the sats that  */
/*               are forecasted by the software  */
/*               to be close to the laser. These */
/*               are not necessarily intersected.*/
/*               ABLPlatform Holds all information about ABL  */
/*               Platform position/disposition  */
/*               ReferenceHour This holds the value of Theta G  */
/*               at RefModJulianDate. The angle  */
/*               of Theta G is given in hours,  */
/*               minutes, and seconds instead of */
/*               degrees, where 24 hrs = 360 deg */
/*               ReferenceMinute Holds the minutes of Theta G at  */
/*               RefModJulianDate.  */
/*               ReferenceSecond Holds the seconds of Theta G at  */
/*               RefModJulianDate.  */
/*               RefModJulianDate This is the reference date when  */
/*               an actual observation of the  */
/*               true value of theta G was made. */
/*               CalcYear Holds the current calender year  */
/*               Calcmonth Holds the Calender month(1 - 12)*/
/*               CalcDay Holds calender day  */
/*               CalcHour Holds the calender hour  */
/*               CalcMinute Holds the calender minute  */
/*               CalcSecond Holds the calender second  */
/*               LazeDuration The amount of time for which the*/
/*               laser will be on. This is  */
/*               To determine how much time in  */
/*               seconds the forecast will last. */
/*               LazerAzimuthInDegrees Lazer Azimuth at Laze Start time*/
/*               in Degrees  */
/*               LazerAzimuthDot The rate of change of the Az  */
/*               in Degrees/Sec.  */
/*               LazerAzimuthDotDot The rate of change of the rate  */
/*               of change of the Azimuth (Accel)*/
/*               in Degrees/Sec^2  */
/*               LazerElevationInDegrees Lazer Elevation at Laze Start  */
/*               in Degrees  */
/*               LazerElevationDot The rate of change of the El  */
/*               in Degrees/Sec.  */
/*               LazerElevationDotDot The rate of change of the rate  */
/*               of change of the Elevat. (Accel)*/
/*               in Degrees/Sec^2  */
/*               SatPositionErrorInMeters Holds the radius of the error  */
/*               spheroid that describes the  */
/*               area in which the satellite is  */
/*               known to exist (in meters).  */
/*               PlatformPositionError...Holds the radius of the error  */
/*               spheroid that describes the  */
/*               area in which the platform is  */
/*               known to exist (in meters).  */
/*               MissilePositionError... Holds the radius of the error  */
/*               spheroid that describes the  */
/*               area in which the missile is  */

```

```

/*          known to exist (in meters).          */
/*          RangeToMissileInKilo... The Range to the missile (km) */
/*          OtherErrorAnglesInDeg Holds any other error angles    */
/*          (in degrees) that may be a          */
/*          significant source of error.          */
/*          This should usually be set to          */
/*          zero (0.0) float.                      */
/*          ThetaGInRadians The angle between the Greenwich */
/*          Meridian and the Vernal Equinox */
/*          at JulianDate.                      */
/*          */
/* OUTPUTS:  NAME:          DESCRIPTION:          */
/*          InFileLength The total number of satellites */
/*          that have been evaluated in the */
/*          InFile          */
/*          OutFileLength The total number of satellites */
/*          that are intersected by platform*/
/*          and have been put in the outfile*/
/*          ClosestApproachFileLength The total number of satellites*/
/*          that come close to the laser */
/*          and have been put in the */
/*          closest approach file.          */
/*          ErrorList Errors that have occurred */
/*          */
/*          THE FINAL OUTPUT IS THE ACTUAL OUTFILE ITSELF WHICH IS */
/*          WRITTEN DIRECTLY TO DISK SO IT CAN BE ACCESSED BY */
/*          OTHER SOFTWARE, IF NEEDED.          */
/*          */
/* COMPILER:  Borland C++ Builder3 Standard version */
/*          This compiler should be used to compile and link. */
/*          */
/*****
PAMainProcessor(char InFileName[MAXNAMELENGTH],
char OutFileName[MAXNAMELENGTH],
char ClosestApproachFileName[MAXNAMELENGTH],
int &InFileLength,
int &OutFileLength,
int &ClosestApproachFileLength,
struct Aircraft &ABLPlatform,
int ReferenceHour,
int ReferenceMinute,
double ReferenceSecond,
double RefModJulianDate,
int CalcYear,
int CalcMonth,
int CalcDay,
int CalcHour,
int CalcMinute,
double CalcSecond,
double LazeDuration,
double LaserAzimuthInDegrees,
double LaserAzimuthDot,
double LaserAzimuthDotDot,
double LaserElevationInDegrees,
double LaserElevationDot,
double LaserElevationDotDot,
double SatPositionErrorInMeters,
double PlatformPositionErrorInMeters,
double MissilePositionErrorInMeters,
double RangeToMissileInKilometers,
double OtherErrorAngleInDeg,
double SecondsFromVertex,
double InterpolationIncrement,

```

```
double &ThetaGInDegrees,  
ErrorStructure &ErrorList);  
  
#endif
```